Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Базовые алгоритмы файлового карвинга

Курсовая работа студента 345 группы
Омельчука Александра Олеговича

Научный руководитель,
ст. преподаватель Ю. А. Губанов

Санкт-Петербург
2010

Оглавление

Введение

Файловый карвинг – это осуществление поиска файлов или записей, основанное на их содержимом, нежели на метаинформации о расположении на носителе.

Файловый карвинг успешно применяется как для восстановления информации, так и в цифровых расследованиях (digital forensic).

Цифровым расследованием называется процесс разработки и проверки гипотез, отвечающих на вопросы о цифровых событиях. Эксперт строит гипотезы на основе найденных улик, затем проверяет их поиском дополнительных улик, которые доказывали бы состоятельность данной гипотезы. Цифровой уликой называется цифровой объект, содержащий надежную информацию, которая поддерживает или опровергает гипотезу. [Кэрриэ, 2007]

К таким уликам, как правило, относятся:

- история посещения веб-страниц
- электронные письма
- истории бесед (Skype, ICQ и пр.)

В случае восстановления информации карвинг может использоваться для восстановления файлов с поврежденных носителей (например, в случае повреждения секторов, содержащих Disk Directory или Master File Table).

Благодаря применению в этих областях, файловый карвинг становится важной задачей.

В настоящее время разработано множество подходов и алгоритмов для осуществления карвинга. Применяемый подход зависит от многих факторов: структуры искомых файлов, содержимого файлов, файловой системы носителя (если она есть).

В данной работе описываются схема работы базового алгоритма файлового карвинга и применение этой схемы для создания более сложных методов цифрового раследования. Также в работе описывается пример внедрения этой схемы в существующий программный продукт.

1. Общие понятия карвинга

В случае криминалистической экспертизы важно исключить любую модификацию исследуемых данных. Для этого данные с диска копируются с использованием устройств, позволяющих только чтение. Копии либо записываются на новый жесткий диск, либо сохраняются в файл-образ. Если проводится копирование оперативной памяти, она также записывается в файл-образ.

Файловый карвинг можно разделить на два типа: базовый и продвинутый.

Базовый карвинг имеет смысл применять при следующих предположениях:

- Начало файла не повреждено
- Файл не фрагментирован
- Файл не сжат (например, сжатие на уровне файловой системы в NTFS)

К базовым типам относятся:

- *Header/Footer Carving* – выделение данных, основанное на известных маркерах начала и конца файла
- *Header/Maximum (file) size Carving* – выделение данных, основанное на известном маркере заголовка и максимальной длине файла
- *Header/Embedded length Carving* – способ, основанный на том, что во многих форматах файлов сразу после маркера начала указана длина записи в том или ином виде.

Продвинутый карвинг применяется в случае фрагментированных файлов:

- Фрагменты могут идти не последовательно
- Порядок фрагментов может быть нарушен
- Некоторые фрагменты могут отсутствовать

К продвинутым типам относятся:

- *File Structure Based Carving* – основан на знании внутренней структуры формата разыскиваемых данных
- *Semantic Carving* – основан на анализе лингвистического либо смыслового содержания данных
- *Carving with Validation* – валидация искомого файла на основании его формата (хороший пример такого подхода описан в [Pereira, 2009]).
- *Fragment Recovery Carving* – данные собираются из разных фрагментов

Способ карвинга также сильно зависит от особенностей файловой системы исследуемого носителя.

Приложения, выполняющие карвинг, принято называть *карверами*.

2. Постановка задачи

Многие существующие бесплатные продукты (scalpel, revit, foremost) предоставляют возможность проведения базового карвинга на основе произвольных сигнатур. При этом некоторые сигнатуры для стандартных форматов файлов (jpeg, zip, и. т. д.) включены в продукт по умолчанию, остальные, в частности, криминалистически интересные форматы истории веб-браузеров или программ мгновенного обмена сообщения (мессенджеров), необходимо описывать самостоятельно. Найденные с помощью базовых алгоритмов результаты не проходят дальнейших проверок и с большой вероятностью могут оказаться некорректными.

В задачу, поставленную перед автором курсовой работы, входит следующее:

- обобщение базовых алгоритмов и разработка универсальной техники поиска структур данных по сигнатурам
- исследование форматов хранения истории популярных веб-браузеров и мессенджеров
- на основе этих форматов – создание продвинутых карверов с эвристиками, позволяющими понизить количество некорректных результатов
- создание синтаксических анализаторов, преобразующих найденные результаты в читаемый для пользователя вид

Форматы хранения разбивают историю на отдельные записи. Во многих случаях эти записи сами имеют отдельные сигнатуры и могут быть найдены с помощью базовых алгоритмов. Полагалось по возможности создавать карверы не для файлов истории целиком, а для отдельных записей, поскольку целые файлы могут иметь большой размер, вследствие чего могут быть фрагментированы или частично повреждены.

Для обнаружения многих форматов достаточно поиска сигнатур. Тем не менее, возможность расширения поиска сигнатур на поиск с помощью масок учитывалась при постановке задачи.

3. Реализация

Ключевыми элементами в реализации предложенного подхода являются:

- Источник данных, реализующий интерфейс IDataSource
- Накопитель данных (класс DataAccumulator)
- Анализатор сигнатур (класс SignatureSearcher)
- Конечный анализатор, наследующий базовый класс Carver

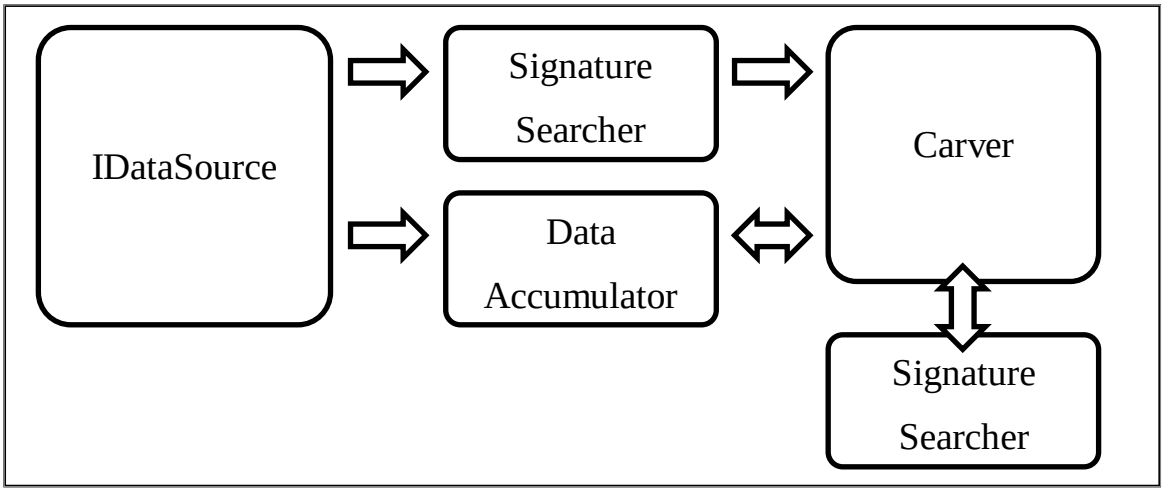

Рисунок 1: Взаимодействие элементов схемы

3.1 Интерфейс IDataSource

Этот интерфейс предоставляет возможность считать сырые данные с источника. Данные возвращаются в виде буферов – байтовых массивов произвольной длины. Разбиение на буфера необходимо, так как источник может быть очень большим (например, образ жесткого диска).

Примеры источников данных:

- *Логический диск.* Данные получаются через WinAPI по запросу CreateFile("\\.\<буква логического диска>:")
- *Физический диск.* Как и в предыдущем случае, данные получаются через WinAPI (запрос CreateFile("\\.\PHYSICALDRIVE<номер диска>"). Этот способ быстрее предыдущего, но приходится читать весь диск.
- *Образ диска или оперативной памяти.*
- *Некоторая сгенерированная структура.* Пример такой структуры будет описан в разделе 5.

3.2 Класс DataAccumulator (накопитель)

Этот класс является прослойкой между активным экземпляром IDataSource и объектами, реализующими логику поиска. Здесь происходит “склеивание” найденных

данных, то есть их разделение на буфера становится прозрачным для карверов. К этому классу обращаются карверы, когда им необходимы дополнительные данные.

Реализация класса-накопителя – чисто техническая задача.

3.3 Класс SignatureSearcher

Как понятно из его названия, данный клас занимается поиском сигнатур в найденном и “склеенном” потоке данных. В случае явно заданных сигнатур эта задача сводится к поиску вхождения шаблона (подстроки) в строке. Требовалось выбрать оптимальный алгоритм поиска с учетом следующих особенностей задачи:

- *Длинная входная строка и короткие шаблоны.* Входной строкой в нашем случае является анализируемый диск или область памяти, и ее длина может достигать нескольких терабайт. Длина сигнатуры вариируется от двух до тридцати символов, то есть ничтожно мала по сравнению с длиной входной строки.
- *Длина входного алфафита равна 256.* Этот параметр важен для оценки времени предварительной обработки данных и работы эвристик, связанных с поиском входного символа в таблицах (для некоторых алгоритмов). Входной алфавит в данной задаче является множеством байтов.
- *Большое количество сигнатур для поиска.* Количество искомых сигнатур не меньше числа карверов в продукте. Планируется иметь карверы истории как всех популярных на данный момент, так и некоторых устаревших мессенджеров, а также карверы оперативной памяти для некоторых мессенджеров и карверы истории популярных веб-браузеров. Таким образом, количество искомых сигнатур превышает 30.
- *“Враждебность” входных данных.* Во входной строке могут содержаться любые данные, например, частые повторения одного и того же символа. Следует учитывать оценку работы алгоритма в худшем случае.
- *Возможность расширить алгоритм на поиск шаблонов с масками.* Эта особенность может понадобиться в будущем при расширении функциональности продукта.

Из первого двух условий следует, что время, затрачиваемое на предварительную обработку данных (осуществляется единожды при инициализации класса) невелико и, следовательно, несущественно при выборе алгоритма. Основную сложность он будет давать при поиске.

Приведем сравнительную таблицу оценки быстродействия наиболее популярных алгоритмов (h – длина входной строки, n – длина искомого шаблона, N – сумма длин всех шаблонов, z – суммарная длина вхождений шаблонов во входную строку):

Алгоритм	Сложность		Множ. шаблоны
	Средняя	Наихудшая	
Автоматный	*O(h)*	*O(h)*	Нет
Кнута-Морриса-Пратта	*O(h)*	*O(h)*	Нет
Бойера-Мура	*O(h)*	*O(hn)*	Нет
Рабина-Карпа	*O(h+n)*	*O(hn)*	Да (одной длины)
Ахо-Корасик	*O(h+N+z)*		Да

Таблица 1: Сравнение основных поисковых алгоритмов

Автоматный алгоритм и алгоритм Ахо-Корасик представляют собой работу конечного автомата, поэтому для их работы фактически не требуется логики склеивания буферов входного текста. Кроме того, они применимы для поиска более сложных шаблонов с масками.

Для алгоритмов, осуществляющих поиск лишь одной строки, сложность следует умножить на количество искомых шаблонов. Для алгоритма Рабина-Карпа сложность следует умножить на количество значений, принимаемых длиной шаблона, для всех шаблонов. Для алгоритма Ахо-Корасик сложность может оказаться квадратичной, если искомые шаблоны вложены друг в друга (например, если в строке “аааа” ищем слова “а”, “аа”, “ааа”). В нашем случае это условие не выполняется; более того, можно утверждать, что *z* много меньше *h*.

По этим соображениям для реализации был выбран алгоритм Ахо-Корасик.

3.4 Класс Carver

Базовый класс Carver представляет собой список начальных сигнатур для поиска, которые он регистрирует в классе SignatureSearcher, и логику обработки события “сигнатура найдена”. При этом он посылает запросы накопителю, когда ему нужны данные.

Рассмотрим пример работы карвера истории Skype 3. Для этого опишем формат, в котором хранятся записи этой истории:

- *Сигнатура – строка “l33l” в формате ASCII*
- *4-байтовое число в формате little-endian – длина записи*
- *Запись*

Соответственно, карвер содержит единственную начальную сигнатуру “l33l”. Когда сигнатура найдена, карвер запрашивает у накопителя следующие 4 байта, преобразует их из формата little-endian в число, и если оно находится в пределах разумного (неотрицательно и меньше максимально допустимого размера записи), запрашивает столько байт у накопителя, и наконец, склеивает все полученные данные и возвращает результат.

Аналогично работают остальные карверы. Они могут использовать различные эвристики (например, распознавать формат xml или sql-записи), индуцированные форматом разбираемого файла. Также карверы могут создавать и использовать свои экземпляры

SignatureSearcher для нахождения сигнатур внутри переданных им данных (так, например, действует Header/Footer carver при поиске сигнатуры Footer). При этом сложная логика включается только по нахождении начальной сигнатуры, поэтому она не сильно влияет на общую производительность.

3.5 Дальнейшая обработка данных

Записи, возвращаемые карвером, удовлетворяют требуемой структуре данных, но это не значит, что они фактически являются искомыми сообщениями. Часть возвращенных записей отвергается синтаксическим анализатором, также для некоторых форматов можно применять дополнительные эвристики. Например, истинность записи сомнительна, если:

- Предполагаемая дата отправки сообщения раньше, чем дата выпуска соответствующего мессенджера (применимо ко многим мессенджерам)
- Строка с сообщением содержит непечатаемые символы, например, символ 0x00
- Адрес страницы в истории веб-браузера не является корректным URL

Эти эвристики включаются в финальный синтаксический анализатор, обрабатывающий результат карвинга для вывода пользователю. Сомнительные записи также сохраняются, чтобы исследователь мог проверить их вручную. Таким образом, при незначительном повреждении записи (например, если ее конец был "затерт" другими данными) возможность обнаружить ее и использовать в цифровом расследовании сохраняется.

3.6 Оптимизация

По результатам исследования производительности прототипа продукта было выяснено, что больше половины времени его работы составляет проход по массиву входных данных в классе SignatureSearcher. Для улучшения времени работы продукта этот класс можно переписать на языке, быстрее работающем с массивами, например, C++.

4. Определение сигнатур

Как упоминалось выше, логика работы карвера определяется форматом структуры, которую он ищет. Поскольку интересные нам мессенджеры зачастую являются закрытыми продуктами, формат их истории также нельзя найти в свободном доступе. Еще сложнее дело обстоит с форматом представления сообщений в RAM. Поэтому для определения каждого формата необходимо проводить исследование соответствующей программы, построив достаточно обширную тестовую историю.

Не все форматы содержат в начале записи отдельные сигнатуры. Например, история веб-браузера Firefox 3.0 хранится в таблице SQLite. Такие записи можно извлечь, зная об их содержимом. Продолжая пример, заметим, что в записи истории веб-браузера всегда встречается имя схемы обращения к ресурсу с соответствующими последующими символами (“http://”, “ftp://”, и т. д.), поэтому за начальную сигнатуру можно принять все подобные строки, и встретив их, запрашивать фиксированное количество байт перед ними и анализировать на соответствие их формату представления записи SQLite-таблицы истории.

Следует также фиксировать версию программы, создавшей тестовую историю, так как формат хранения данных (и тем более их представления в оперативной памяти) может меняться от версии к версии.

Ниже приведена таблица форматов, для которых построены карверы:

Формат	Версия
Skype 3	-
Skype 4	4.2.0.158
Digsby	r27209 (build 78)
ICQ Lite	-
Yahoo! Messenger	-
Miranda IM	0.8.10
Windows Live Messenger	14.0.8089.726
QIP Infium	9034, QIP 2010
Windows Live Messenger RAM	14.0.8089.726
Skype RAM	4.2.0.158
ICQ 7 RAM	7.1.0.2096
Yahoo! Messenger RAM	-
Gmail RAM	-
Internet Explorer	6, 7, 8
Firefox 3	3.0

Таблица 2: Поддерживаемые карверы

5. Применение базового алгоритма

Предложенный подход неплохо справляется с источниками, в которых файлы хранятся в нетронутом виде. Однако многие файловые системы каким-либо образом изменяют файл. Например, в файловой системе NTFS длинные файлы разбиваются на несколько частей, которые могут быть записаны в различные места на диске. В той же файловой системе существует возможность сжимать файлы при записи на диск для экономии места. Изменения, проделанные файловой системой, вместе с остальной информацией о файлах и каталогах хранятся в главной файловой таблице (Master File Table, MFT). Таким образом, при восстановлении этой таблицы можно получить можно дополнительной информации для работы карвера.

На практике был применен метод двухпроходного карвинга:

- К содержимому NTFS-диска, помимо прочих, применяется MFT-карвер.
- На основе восстановленных записей файловой таблицы создается класс FragmentedDataSource. Он разжимает фрагменты файлов с помощью того же алгоритма, что используется драйвером NTFS, и упорядочивает их. Этот класс наследует интерфейс IDataSource.
- К полученному источнику данных применяется стандартная схема карвинга.

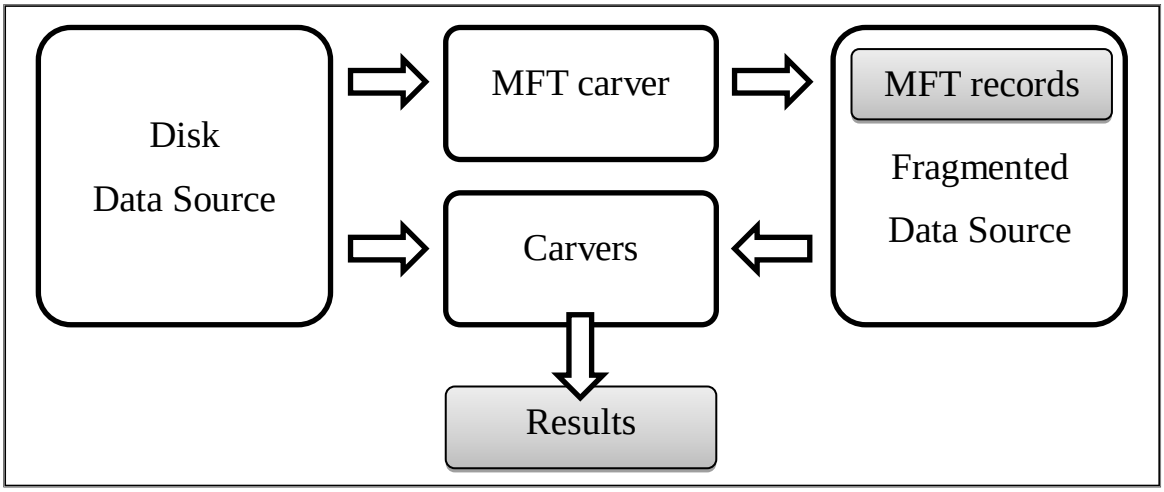

Рисунок 2: Схема двухпроходного карвинга

Описанный метод способен находить больше записей, чем применение стандартного NTFS-драйвера, поскольку он “не брезгует” записями, помеченными файловой системой как недействительные.

Подобные расширения базового алгоритма можно применять и для других файловых систем, изменяющих данные, если уметь строить для них соответствующие экземпляры IDataSource.

Заключение

В рамках курсовой работы была рассмотрена и изучена проблема файлового карвинга. Была предложена эффективная и расширяемая схема работы карвера, послужившая основой для более сложного метода поиска. Также были определены форматы истории некоторых программ мгновенного обмена сообщениями.

Описанный подход применен в существующем программном продукте.

Список литературы

1. Кэрриэ, Б. (2007). *Криминалистический анализ файловых систем*. Издательство "Питер".
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. (2001). *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill
3. Pereira, M. T. (2009). *Forensic Analysis of the Firefox 3 Internet History and Recovery*. Digital Investigation Journal, May , 93–103.
4. Keith. J. Jones. (2003). *Forensic Analysis of Internet Explorer Activity Files.*
5. Detective Eric Oldenburg. (б.д.). *Searching for Yahoo Chat fragments in Unallocated Space*. Phoenix Police Department.