Санкт-Петербургский государственный университет

Киреев Илья Владимирович

Выпускная квалификационная работа

Криминалистический анализ файловой системы Btrfs

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование информационных систем»*

Основная образовательная программа *СВ.5006.2018 «Математическое обеспечение и администрирование информационных систем»*

Профиль *Системное программирование*

Научный руководитель:
доцент кафедры СП, к.т.н. Ю.В. Литвинов

Рецензент:
Ведущий инженер-программист ООО «Софтком» А.Р.Ханов

Санкт-Петербург
2022

Saint Petersburg State University

Kireev Ilya Vladimirovich

Bachelor's Thesis

Forensic analysis of the Btrfs file system

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *"Software and Administration of Information Systems"*

Profile: *System Engineering*

Scientific supervisor:
Docent, C.Sc. Yurii Litvinov

Reviewer:
Lead software engineer LLC «Softcom» Artur Khanov

Saint Petersburg
2022

Оглавление

Введение

В настоящее время совершается все больше преступлений в сфере информационных технологий, для их расследования существует специальная отрасль судебной науки — цифровая криминалистика. Она изучает извлечение и исследование данных, обнаруженных на цифровых устройствах. В ходе своей работы эксперты-криминалисты часто сталкиваются с необходимостью анализа данных с жестких дисков и переносных носителей, что в свою очередь требует подробного изучения файловой системы, используемой на устройстве.

Операционные системы семейства Unix[1], помимо серверных решений, начинают набирать популярность также и среди рядовых пользователей. В их основе лежит базовое монолитное ядро, а дополнительные компоненты, такие как драйверы для новых устройств или файловые системы (ФС), могут быть скомпилированы отдельно. По этой причине для инструментов цифрового криминалистического анализа крайне необходима поддержка ряда специфических файловых систем, в частности Btrfs [2].

Btrfs — это файловая система, основанная на принципе «Copy on Write» (CoW), который заключается в сохранении измененных данных в новое свободное пространство на диске, без модификации оригинала, что позволяет избежать их повреждения или частичной записи в случае сбоя питания во время внесения изменений. Также этот принцип позволяет делать эффективные снимки[2] системы, которые не занимают места на диске до внесения изменений в файлы. С 2009 года Btrfs доступна в ядре Linux, а в ноябре 2013 года она получила статус стабильной и некоторые дистрибутивы (Ubuntu, Arch, Fedora) позволяют установить ее в качестве основной. Такие компании, как например Suse в «Linux Enterprise Server» [4] или NetGear в «ReadyNas» [5], активно используют Btrfs в своих продуктах.

Компания «Белкасофт» занимается разработкой программного обес-

[1]Unix: https://opengroup.org/unix (online, accessed: 27.05.2022)

[2]Снимок файловой системы — копия файлов и директорий и определенный момент времени

печения для цифровой криминалистики, одним из ее продуктов является Belkasoft Evidence Center X[3] для ОС Windows. Инструмент позволяет проводить анализ данных для экспертизы из разнообразных источников, в том числе с жестких дисков, для которого необходимо получение максимально возможного количества артефактов.

Проведение анализа средствами операционной системы не позволяет исследовать поврежденные данные, поскольку операции с ними могут нарушить работоспособность всей файловой системы, что недопустимо для ОС. В контексте криминалистического анализа мы этим не ограничены, поскольку работаем с данными только в режиме чтения, а для исследования важны любые доступные артефакты. По этой причине оригинальный модуль для Linux и популярный open-source[4] драйвер WinBtrfs[5] для Windows не подходят за основу для исследования.

Популярный инструмент с открытым исходным кодом для криминалистического анализа The Sleuth Kit (TSK)[6] в актуальной версии Btrfs не поддерживает. По этой причине было принято решение самостоятельно изучить возможности извлечения поврежденных и удаленных данных для актуальной версии файловой системы, в том числе для RAID конфигураций и внедрить их в продукт Belkasoft Evidence Center X компании Belkasoft.

[3]Belkasoft Evidence Center X: https://belkasoft.com/x (online, accessed: 27.05.2022)

[4]Open-source software — программное обеспечение с открытым исходным кодом

[5]WinBtrfs: https://github.com/maharmstone/btrfs (online, accessed: 27.05.2022)

[6]https://github.com/sleuthkit/sleuthkit (online, accessed: 27.05.2022)

1. Постановка задачи

Целью данной работы является изучение и применение возможностей извлечения поврежденных и удаленных данных в файловой системе Btrfs с апробацией предложенных механизмов в коммерческом продукте компании «Belkasoft». Для ее достижения были поставлены следующие задачи.

- Провести обзор существующих решений.
- Исследовать возможности восстановления удаленных и поврежденных данных в Btrfs.
- Реализовать модуль для криминалистического анализа файловой системы Btrfs с возможностью восстановления удаленных и поврежденных данных.
- Интегрировать разработанный модуль в продукт Belkasoft Evidence Center X.

2. Обзор

2.1. Существующие решения для криминалистического анализа Btrfs

2.1.1. Проприетарные решения

Среди коммерческих продуктов, которые позволяют проводить криминалистический анализ файловых систем можно выделить наиболее популярный «X–Ways Forensics»[7] от X–Ways, который также поддерживает Btrfs. Среди возможностей данного инструмента заявляется анализ подтомов, снимков, анализ RAID конфигураций из Linux, поддержка нескольких устройств, встроенная в Btrfs, пока недоступна. Также указывается возможность поиска удаленных файлов, в том числе частично поврежденных, определение отсутствующих устройств в томе. Для решений от компаний Magnet [3], AccessData и enCase, Btrfs не заявляется как поддерживаемая.

Исходя из найденных возможностей инструмента X–Ways, подтверждается предположение о возможности восстановления удаленных и поврежденных данных в Btrfs.

2.1.2. WinBtrfs

Существует open–source драйвер WinBtrfs[8], который позволяет работать с Btrfs в операционной системе Windows и к тому же пользуется большой популярностью. Но эксперту–криминалисту порой приходится работать с системами, не имея полного доступа к ним, например в случае использования в системе BitLocker[9] дисков не зная пароля администратора. В таком контексте установить драйвер не представляется возможным, к тому же возникающие критические ошибки в его работе приведут к полной перезагрузке и утрате доступа к ним до ввода

[7]https://www.x-ways.net/forensics/index-m.html (online, accessed: 27.05.2022)

[8]https://github.com/maharmstone/btrfs (online, accessed: 27.05.2022)

[9]https://docs.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-overview (online, accessed: 27.05.2022)

пароля. Но основной причиной является невозможность исследовать поврежденную систему средствами ОС.

2.1.3. The Sleuth Kit

Наиболее популярным open–source инструментом для проведения криминалистического анализа является The Sleuth Kit (TSK). Он реализован в виде библиотеки и поддерживается, в том числе на Windows. Для него существует две различных реализации с поддержкой Btrfs: одна[10] не обновляется уже 8 лет, вторая[11] — 4 года и обе не добавлены в основную ветку библиотеки. При сборке данных решений возникает множество ошибок, а чтение образа актуальной версии файловой системы завершается неудачно. Они могут быть использованы в качестве основы для исследования, поскольку основные структуры системы остались неизменны.

2.2. Устройство файловой системы

2.2.1. Суперблок

Btrfs начинается с «суперблока» — структуры, которая хранит основную информацию о системе, в том числе: логические адреса корневого дерева и дерева чанков, количество устройств, использующихся в конфигурации, и системные блоки, которые позволяют произвести первоначальную адресацию логических адресов в физические. Для обеспечения отказоустойчивости файловая система хранит несколько (в зависимости от размера устройства) копий суперблока, которые обновляются одновременно.

2.2.2. Деревья

Вся система строится на B-деревьях [6], они хранятся в виде заголовка и узла корня. Адреса заголовков хранит корневое дерево, в свою

[10]https://github.com/sleuthkit/sleuthkit/pull/413 (online, accessed: 27.05.2022)
[11]https://github.com/fkie-cad/sleuthkit (online, accessed: 27.05.2022)

очередь адрес его собственный адрес находится в суперблоке. Все данные в файловой системе хранятся в листьях дерева, а промежуточные узлы содержат только ссылки на потомков. Рассмотрим те, которые нам понадобятся для чтения системы.

- Root tree (Корневое дерево) — хранит адреса корней остальных деревьев, в том числе подтомов и снимков. Ссылка на корень root дерева хранится в суперблоке.

- Chunk tree (Дерево чанков) — выполняет преобразование логического адреса в один или несколько физических, в зависимости от конфигурации тома. В Btrfs везде используются логические адреса, в том числе и у дерева чанков, поэтому суперблок содержит чанк с адресом корня этого дерева для начального сопоставления. Также тут хранится информация об используемых устройствах.

- Filesystem tree — представляет один подтом системы, храня информацию о всех директориях и файлах входящих в него, в том числе метаданные экстентов[12] с данными. Первый подтом, имеющий идентификатор 5, представляет корневую директорию основного тома, следующий имеет 256, 257 и так далее.

2.2.3. RAID

Btrfs позволяет объединять внутри одной файловой системы несколько устройств и работать с общим пулом хранения. При объединении нескольких дисков в общий том есть возможность использования следующих RAID[13] конфигураций[14].

- RAID 0 — все диски в массиве представляются как единое пространство, информация делится на равные блоки и поочередно

[12]Экстент — непрерывная область хранения, зарезервированная для файла.

[13]RAID — Redundant Array of Independent Disks

[14]https://btrfs.wiki.kernel.org/index.php/Using_Btrfs_with_Multiple_Devices (online, accessed: 27.05.2022)

пишется на устройства в массиве, что позволяет достичь наибольшей скорости чтения/записи. В случае повреждения одного устройства, все данные будут потеряны.

- RAID 1 («зеркалирование») — данные дублируются на все устройства в массиве. Скорость записи не меняется относительно одного устройства, а при чтении скорость возрастает кратно числу устройств в массиве, поскольку выполняется поочередно. Допускается потеря всех устройств, кроме одного.

- RAID 5 — блоки данных и их контрольные суммы записываются циклически на все устройства в массиве. Для вычисления контрольной суммы используется операция XOR[15], для хранения которой суммарно используется только один диск. Система будет работать даже при повреждении одного из устройств. Для создания необходимо как минимум 3 диска.

- RAID 6 — используется циклическая запись, аналогично RAID 5, с одним дополнительным томом четности, которые обычно вычисляются с помощью кодов Рида–Соломона [7]. Система будет работать даже при утрате сразу двух устройств. Для создания понадобится как минимум 4 диска.

- RAID 10 (1 + 0) — Представляет собой объединение RAID–1 массивов в RAID–0 массив. Данные могут быть утрачены только в случае выхода из стоя всех устройств в одном RAID–1 массиве. Необходимо минимум 4 устройства.

Реализации RAID 5 и RAID 6 для BTRFS на момент написания работы имеют статус «нестабильных».

2.2.4. Подтома и снимки

Btrfs поддерживает создание подтомов — каталогов, которые имеют возможность монтирования и размонтирования, со своими правами

[15]Исключающее «или»

и опциями, по сути отдельных файловых систем. Они реализованы на уровне файловой системы, а не поверх нее, в отличие от, например, томов LVM. Подтома разделяют все свободное пространство системы между собой, без необходимости указания фиксированного размера для каждого, также они могут разделять между собой общие данные, избегая их копирования.

Снимки — специальный тип подтомов, который представляет собой состояние некоторого подтома в определенный момент времени. Благодаря принципу CoW, они занимают очень мало места, так как не выполняется дублирование, а хранится только дельта измененных данных. Снимки могут быть использованы как резервная копия, создавая их с модификатором read–only. С ними можно работать и как с обычными подтомами, в том числе монтировать и модифицировать данные.

3. Исследование возможностей восстановления удаленных и поврежденных данных в Btrfs

3.1. Восстановление суперблока

Разбор системы невозможен без получения корректного суперблока. Как уже упоминалось ранее, на диске хранится несколько его копий, основной имеет смещение 0x10000 (64 KB), а дубликаты: 0x4000000 (64 MB), 0x4000000000 (256 GB) и так далее, если размер устройства это позволяет. Если первый суперблок окажется поврежден, то система даже не будет проверять остальные и BTRFS не сможет быть смонтирована, хотя ее все еще можно легко прочитать.

3.2. Логический адрес в физический

В BTRFS всюду используются логические адреса, такое решение позволяет существенно упростить адресацию в пуле хранения с несколькими устройствами. Для преобразования их в физические адреса на устройстве необходимо для начала найти чанк, в котором находится искомый адрес.

Чанк представляет собой непрерывный участок логической памяти, он хранит массив полос, который представляет собой непрерывный участок памяти уже на устройстве. Таких массивов может быть несколько, если используется дублирование, как например для метаданных, или выбрана соответствующая RAID–конфигурация. В таком случае при ошибке чтения одной полосы или, например, отсутствии одного из устройств, мы можем прочитать ее дубликат. Если невозможно прочитать ни одну из копий полосы данных файла, мы можем заполнить этот блок нулями, что позволит сохранить общую структуру документа и ознакомиться хотя бы с частью артефактов. Такой подход, к сожалению, не подойдет в случае поврежденных метаданных, поскольку они существенны для дальнейшего разбора системы.

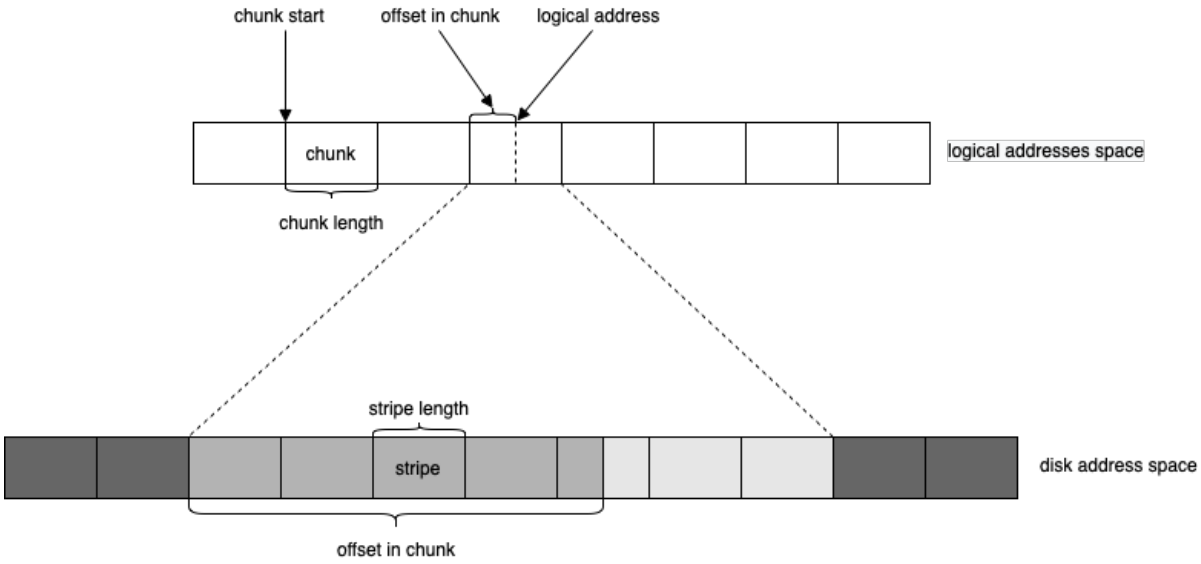

Рис. 1: Преобразование логического адреса в физический

Чанк для первоначальной адресации основных деревьев хранится в системных блоках в суперблоке. Дальнейшее преобразование логических адресов в физические выполняется с помощью дерева чанков.

Полосы хранят идентификатор устройства, на котором они находится, и смещение на нем для чтения данных. Они не существуют отдельно от чанков, которые также хранят их длину и количество в массиве.

3.3. Восстановление метаданных директорий и файлов

Основной задачей при разборе файловой системы является получение метаданных файлов и директорий, по которым затем читаются данные. Они хранятся в структурах dir_item, содержат идентификатор узла, его тип, имя и с хэшем. По–умолчанию для хэширования используется CRC32C[16], но при создании системы могут быть выбраны еще и следующие типы: XXHASH[17], SHA256[18], BLAKE2b[19], выбранный тип хранится в суперблоке. В случае ошибки проверки хэша, мы можем вос-

[16]https://github.com/google/crc32c (online, accessed: 27.05.2022)
[17]https://github.com/Cyan4973/xxHash (online, accessed: 27.05.2022)
[18]https://datatracker.ietf.org/doc/html/rfc4634 (online, accessed: 27.05.2022)
[19]https://www.blake2.net (online, accessed: 27.05.2022)

становить имя и тип из структуры dir_index, которая по содержанию дублирует первую, а в inode_ref еще раз дублируется имя. При добавлении нового файла в систему, структуры метаданных добавляются в деревья пустыми и затем последовательно заполняются данными, а значит отсутствующие или поврежденные данные в одних структурах возможно получить из других.

3.4. Копии метаданных

Суперблок, помимо ранее упомянутых данных, хранит 4 предыдущих версии метаданных системы, в них доступна следующая информация.

- Адреса предыдущих версий корневого дерева и дерева чанков, номер их поколения и высоту
- Максимальный размер системы
- Количество использованных байт
- Количество устройств в пуле

В случае повреждения дерева чанков или корневого дерева с актуальными данными, можно восстановить их предыдущую версию. При возникновении ошибок в процессе инициализации дерева файловой системы или одного из подтомов из актуального корневого дерева чтение их из предыдущих генераций деревьев возможно позволит получить хотя бы часть данных. Если же крайняя модификация доступна, чтение ранних версий позволит получить удаленные данные и файлы. Новая копия метаданных создается раз в 30 секунд [1], что позволит восстановить данные, удаленные как минимум 2 минуты назад.

3.5. Определение отсутствующих устройств в пуле

С помощью информации в суперблоке, копиях метаданных и дереве чанков можно восстановить данные о недостающих устройствах в

пуле в случае неполного RAID–массива. Они хранятся в структурах dev_item, которые предоставляют следующую информацию: количество используемых байт на устройстве, общий размер диска, идентификатор диска, уникальный для каждого девайса в системе, размер сектора, смещение раздела с Btrfs и id файловой системы.

После получения первого снимка для анализа, прочитав в суперблоке число устройств в системе и сверив его с количеством элементов dev_item в дереве чанков, для дополнительной проверки корректности данных, мы можем запросить у пользователя добавить недостающие снимки дисков, в случае их наличия. При добавлении нового снимка к анализу, после определения существования на нем суперблока, по id системы в структуре устройства можно проверить принадлежность его к текущей системе, а идентификатор устройства позволит удостовериться, что он не является дубликатом уже добавленных.

3.6. Orphan элементы

Существует особый тип orphan, он используется для элементов, которые находятся в процессе удаления. Данный процесс выполняется в несколько этапов и может быть прерван в случае сбоя питания или ошибки системы, чтобы система могла быть восстановлена, удаляемому элементу присваивается специальный тип. После инициализации системы такие элементы подлежат удалению, в случае обнаружения их в деревьях они позволят найти дополнительные артефакты, которые могут быть удаленными файлами или целыми директориями, недоступными в актуальной системе.

4. Реализация

4.1. Чтение деревьев

После получения логического адреса заголовка корня необходимого дерева, например из суперблока или уже инициализированных ранее деревьев, выполняется чтение узла корня. Заголовок содержит информацию об уровне узла и количестве элементов в нем, а сразу после хранится массив структур, так например в случае листа это массив заголовков элементов, как представлено на Рис. 2. Если уровень узла равняется нулю, то это лист.

Рис. 2: Хранение листа дерева в логическом адресном пространстве

Заголовок хранит смещение данных относительно конца заголовка узла и размер данных, а также ключ, который предоставляет следующую информацию об элементе:

```
struct btrfs_key
{
     uint64_t id;
     item_type type;
     uint64_t data;
}
```

Поле data хранит специфичную для каждого типа элементов информацию, так например для dir_item там находится контрольная сумма имени, а для dir_index порядок в директории. Для данных типов элементов поле id в ключе хранит идентификатор родителя, а не текущего узла, по которому все элементы отсортированы для оптимизации поиска. Номер самого элемента хранится внутри самой структуры элемента.

Если же уровень в заголовке больше нуля, то этот узел является промежуточным, он хранит после себя массив структур key_ptr. Они

представляют краткую информацию о потомке, такую как минимальный id, существующий в данном поддереве, что позволяет существенно оптимизировать поиск элемента по идентификатору. Хранение узла в памяти устроено следующим образом:

Рис. 3: Хранение промежуточного узла дерева в логическом адресном пространстве

Для оптимизации на данном этапе можно не выполнять автоматически дальнейшее чтение потомков пока пользователь не захочет открыть хранящиеся там данные, но такой подход не подойдет для продукта Evidence Center X, поскольку он требует предварительно чтения всей системы для поиска артефактов, но поскольку мы работаем с деревьями, чтение каждого key_ptr можно выполнять в отдельном потоке.

4.2. Чтение элементов деревьев

Типы элементов листьев реализованы как представлено на Рис. 4. Они наследуются от абстрактного класса item, который хранит заголовок элемента. Необходимый тип выбирается после чтения заголовка в зависимости от типа, хранящегося в ключе.

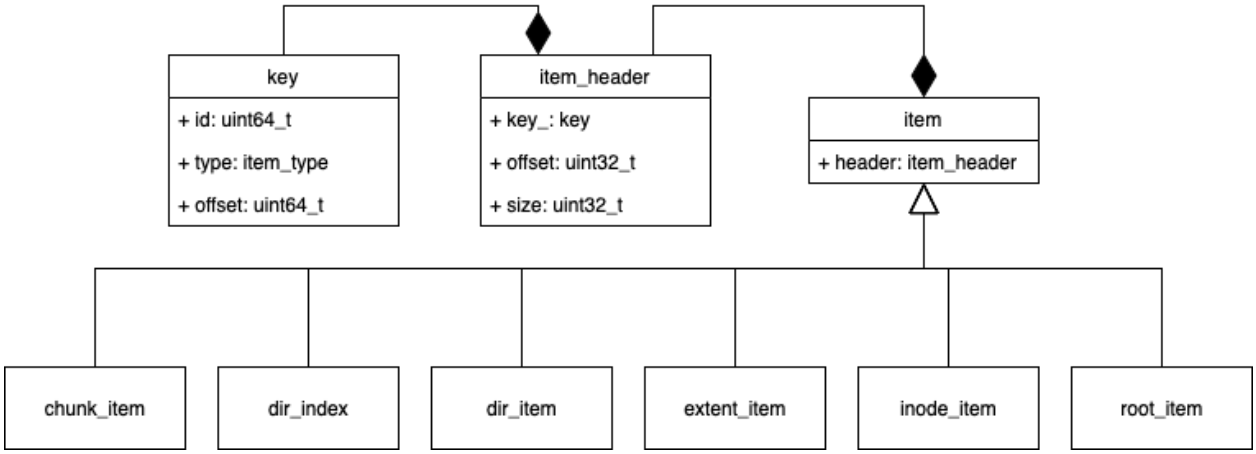

Рис. 4: Устройство элементов деревьев

4.3. Чтение директорий и файлов

Каждый хранящийся в дереве файловой системе идентификатор отвечает либо за файл, либо за директорию, у которых для каждой существует структура dir_item, она хранит следующие данные:

```
struct dir_item
{
     btrfs_key key;
     uint64_t transaction_id;
     uint16_t data_len;
     uint16_t name_len;
     dir_item_type type;
     char* name;
}
```

Поскольку ключ в заголовке этого элемента хранит идентификатор родителя, другой, хранящий информацию о текущем узле, существует внутри самой структуры. В случае если он указывает на тип inode_item, то элемент находится в текущем томе, а если root_item, то элемент является подтомом. Поле dir_item_type хранит тип элемента в директории: файл, директория, блочное устройство, сокет или символьная ссылка.

4.4. Чтение данных файлов

Для файлов используется несколько способов хранения данных, так если данные не превосходят максимальный размер[20] для inline файлов, то данные хранятся в элементе сразу после общей информации в extent_item. Для более крупных файлов данные хранятся независимо от метаданных, для одного файла может использоваться несколько таких структур, если файл превосходит размер одного экстента, каждый элемент хранит его логический адрес.

[20]Максимальный размер узла (по–умолчанию 4 КБ) минус размер используемых метаданных для его хранения

Каждый extent_item может использовать свой тип сжатия и шифрования, который хранится внутри структуры, как и размер исходных данных. Для сжатия данных Btrfs поддерживает ZLIB[21], LZO[22] и ZSTD[23], а шифрование производится только на уровне операционной системы, интеграция fscrypt[24], для шифрования на уровне файлов и директорий сейчас находится в разработке.

4.5. Чтение подтомов и снимков

Снимки и подтома хранятся в системе одинаково, поскольку первые являются частным случаем вторых. Для каждого подтома в дереве корневой системы содержится root_item, представляющий корень файловой системы для соответствующего подтома, и root_ref, с помощью которого можно найти том верхнего уровня. В случае повреждения или ошибки чтения root_ref так также хранятся элементы root_backref, которые хранят идентификаторы текущего узла и родителя в обратном порядке.

Дерево файловой системы является томом самого верхнего уровня, оно представляет корневую директорию с именем пути «/», а первый созданный подтом будет иметь id 256 с идентификатором родителя 5. Элементы dir_item внутри себя содержат дополнительный ключ, который имеет тип inode_item, если элемент является обычным файлом или директорией, либо root_item, если это подтом. Нумерация подтомов ведется независимо от остальных элементов, так внутри одной директории может существовать, например, файл и снимок с id 257 и отличить их в таком случае без дополнительной информации невозможно.

Продукт Belkasoft Evidence Center X, в который предстояло в дальнейшем интегрировать разработанное решение, использует для итерации по директориям и файлам только их идентификаторы, но при исследовании Btrfs с подтомами работать с ними в стандартном виде

[21]https://zlib.net (online, accessed: 27.05.2022)
[22]https://fossies.org/linux/lzo/doc/LZOAPI.TXT (online, accessed: 27.05.2022)
[23]https://github.com/facebook/zstd (online, accessed: 27.05.2022)
[24]https://wiki.archlinux.org/title/Fscrypt (online, accessed: 27.05.2022)

невозможно, ввиду их множественного дублирования. В связи с этим было принято решение использовать первый байт id для хранения номера подтома, что позволит однозначно определять элемент только по его идентификатору, существенно не уменьшая максимальное количество файлов для исследования. Для Btrfs максимальное число файлов и директорий равняется 2^{64}, для подтомов нумерация ведется отдельно и также ограничивается 64 битами. Внесенная модификация позволит исследовать систему не более с чем 2^{56} файлами и директориями и 256 подтомами, чего будет достаточно для исследования большинства систем и в дальнейшем, в случае запросов от пользователей, количество байт под номер подтома может быть увеличено.

Был реализован класс Subvolume, который инициализируется корнем дерева файловой системы соответствующего подтома и списком корней вложенных в него подтомов. Индекс подтома по идентификатору корня вычисляется как `id XOR 256ULL + 1`, затем с помощью битового сдвига он добавляется в старший байт id файла или директории. Обратные преобразования выполняются аналогично, а благодаря использованию битовых операций, они эффективны по скорости вычисления. При обнаружении в дереве файловой системы очередного корня подтома, вычисляется его индекс по идентификатору, если среди потомков есть подтом с таким индексом, то чтение выполняется из него, иначе оставляем его обычной пустой директорией.

5. Интеграция

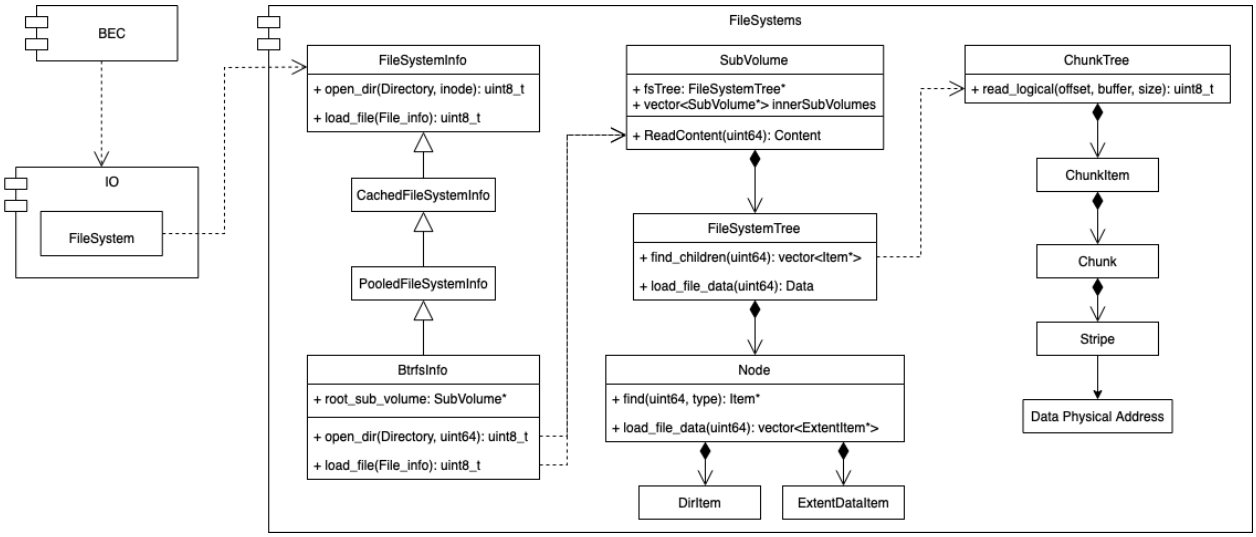

Рис. 5: Архитектура решения по исследованию файловой системы Btrfs в Belkasoft Evidence Center

Реализованное решение было интегрировано в Belkasoft Evidence Center X. В библиотеку FileSystems, которая выполняет чтение и разбор различных файловых систем, была добавлена поддержка нового типа — Btrfs. Был реализован абстрактный класс PooledFileSystemInfo, который используется для работы с файловыми системами с пулом хранения, в частности Btrfs. Данный класс позволяет использовать для анализа одной системы несколько образов, выполняя сопоставление логического адреса устройству, с которого возможно прочитать данные. Интерфейс CachedFileSystem использует базу данных для сохранения промежуточных результатов разбора системы, который позволяет избежать повторного анализа уже исследованных ранее данных. Интерфейс FileSystemInfo позволяет выполнять поиск файлов и директорий в загруженной системе по его идентификатору.

Метод Open в классе BtrfsInfo, выполнив предварительную проверку существования суперблока на переданном образе, инициализирует все необходимые деревья в системе, строит дерево подтомов и заполняет основную информацию о системе, такую как, например, размер полос, количество файлов и директорий, количество устройств в пуле.

Метод open_dir используется как для файлов, так и для директо-

рий, он позволяет определить по id существует ли такой элемент, чем он является и загрузить все данные о нем: список потомков в случае директории или данные в случае файла.

Модуль IO использует библиотеку FileSystems и в класс FileSystem тип файловой системы Btrfs был добавлен как возможный в результате анализа, а BEC реализует пользовательский интерфейс и высокоуровневую логику программного продукта.

Заключение

В ходе работы были достигнуты следующие результаты.

- Проведен обзор существующих решений.
- Исследованы возможности восстановления удаленных и поврежденных данных в Btrfs.
- Реализован модуль для криминалистического анализа файловой системы Btrfs с возможностью восстановления удаленных и поврежденных данных.
- Разработанный модуль интегрирован в продукт Belkasoft Evidence Center X.

Список литературы

[1] Btrfs Arch Linux Doc. — online; accessed: 27.05.2022. URL: https://wiki.archlinux.org/title/btrfs.

[2] Btrfs wiki. — online; accessed: 27.05.2022. URL: https://btrfs.wiki.kernel.org/index.php/Main_Page.

[3] Magnet «Axiom» user-guide. — online; accessed: 27.05.2022. URL: http://092f67184f02fcdb918c-b3d937de523d4a3d4cea730efa685a0d.r37.cf1.rackcdn.com/AXIOM%20docs/Magnet%20AXIOM%20User%20Guide.pdf.

[4] Overview of File Systems in Linux. — online; accessed: 27.05.2022. URL: https://documentation.suse.com/sles/15-SP1/html/SLES-all/cha-filesystems.html.

[5] ReadyNAS® Introduces the Next Generation of Storage. — 2013. — online; accessed: 27.05.2022. URL: https://www.netgear.com/media/BTRFS%20on%20ReadyNAS%20OS%206_9May1318-76105_tcm148-107253.pdf.

[6] Shetty Soumya B. A user configurable implementation of B-trees. — 2010. — online; accessed: 27.05.2022. URL: https://dr.lib.iastate.edu/handle/20.500.12876/25512/.

[7] Р. Блейхут. Теория и практика кодов, контролирующих ошибки. — 1986. — С. 576. — online; accessed: 27.05.2022.