СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Обеспечение защиты от неправомерного доступа — мера по защите информации. При работе с информацией лицо, имеющее доступ к этой информации, ограничено правами, которые определяют способы взаимодействия с информацией. В качестве примера таких прав можно привести права на чтение, изменение и удаление информации. Возможность доказательства неправомерного доступа — способность системы хранения данных реагировать на изменения в данных, происходящих не через эту систему, и сигнализировать об этом при запросе.

Цель работы — реализовать улучшение метода блочного хранения данных в СУБД ClickHouse в движке MergeTree, дающее возможность доказательства неправомерного доступа.

Для достижения поставленной цели требуется решить следующие задачи:

— описать инструменты и технологии, требуемые для реализации нового метода;
— реализовать новый метод в виде класса, отвечающего за управление цепью хеш-сумм, связывающую куски данных;
— встроить полученную реализацию класса в функционал движка MergeTree;
— исследовать метод на предмет возможности реализации угроз (обхода защиты) при нешифрованном хранении данных;
— исследовать корректность работы метода при различных событиях системы при шифрованном хранении данных.

1 Технологическая часть

В данном разделе рассматриваются используемые инструменты и технологии для реализации описанного метода, а также сама реализация.

1.1 Выбор инструментов и технологий

В качестве языка программирования был выбран C++ [1] в связи с тем, что на этом языке программирования написана СУБД ClickHouse, для которой данный метод реализуется.

В качестве компилятора был выбран компилятор Clang [3] 12 версии в связи с тем, что на официальном сайте ClickHouse рекомендуется использовать Clang выше 11 версии [2].

Для сборки проекта используются утилиты CMake [4] и Ninja [5], в проекте уже присутствуют требуемые конфигурационные файлы и сборка полностью автоматизирована.

1.2 Код объекта, отвечающего за построение цепи хеш-сумм

Для проверки и расчета цепи хеш-сумм используется класс `MergeTreeDataChainer`. Его интерфейс представлен в листинге 1. Публичный интерфейс состоит из следующих методов:

1) `precommitChain` — проверяет, что текущая цепь хеш-сумм валидна и записывает в журнал значение новой. В качестве аргументов получает текущую цепь, элемент, который добавляется в цепь, вектор элементов, которые удаляются из цепи, а также блокировку на использование кусков. Возвращает булево значение, было ли записано значение в журнал (не записывается в случае неверной цепочки на момент вызова метода).
2) `checkConsistency` — проверяет цепь. На вход получает набор кусков, а также блокировку на использование кусков.
3) `commitChain` — фиксирует значение цепи хеш-сумм, перенося его из журнала в файл цепи. На вход получает блокировку на использование кусков.

Листинг 1: Класс MergeTreeDataChainer.

```
class MergeTreeDataChainer {
    using Checksum = UInt64;
    using Checksums = std::vector<Checksum>;

public:
    MergeTreeDataChainer(DiskPtr disk, String relative_storage_path,
        Poco::Logger *log = nullptr);
    bool precommitChain(DataParts &data_parts, const
        DataPartPtr &part_to_add,
        const DataPartsVector &parts_to_remove,
        const DataPartsLock &lock);
    CheckResult checkConsistency(const DataParts &data_parts,
        const DataPartsLock & /*lock*/);
    void commitChain(const DataPartsLock & /*lock*/);

private:
    Checksums calculateChain(const DataParts &data_parts);
    std::optional<String> compareChains(const Checksums &checksums,
        const Checksums &written, const DataParts &data_parts);
    static void transformToFutureState(DataParts &data_parts,
        const DataPartPtr &part_to_add,
        const DataPartsVector &parts_to_remove);
    CommitFinisherPtr precommitChain(const DataParts &data_parts,
        const DataPartsLock & /*lock*/);
    void updateFromOnePart(SipHash &hash, const DataPart &data_parts);

    State state;
    Poco::Logger *log;
};
```

Метод, отвечающий за проверки корректности цепи хеш-сумм, представлен в листинге 2. Данный код сначала проверяет соответствие цепочки зафиксированному в файле значению. Если обнаруживается несовпадение, то результат сравнивается с значением из журнала. Если не совпадает со значением из журнала, то возвращается ошибка.

Листинг 2: Валидация цепи хеш-сумм.

```
CheckResult MergeTreeData::MergeTreeDataChainer::checkConsistency(
        const DataParts &data_parts, const DataPartsLock & /*lock*/) {
    const auto checksums = calculateChain(data_parts);
    const auto commited = state.readCommited();
    auto error = compareChains(checksums, commited, data_parts);
    if (!error)
        return CheckResult("parts chain", true, "");

    const auto pending = state.readPending();
    error = compareChains(checksums, pending, data_parts);
    if (!error) {
        if (log)
            log->warning(
                    "parts chain is verified with pending value,
   rewriting commited");
        state.writeCommited(checksums);
        return CheckResult("parts chain", true, "");
    }
    return CheckResult("parts chain", false, std::move(*error));
}
```

Сравнение цепей хеш-сумм сводится к сравнению 2 векторов. Код, рассчитывающий значение цепи хеш-сумм, представлен в листинге 3. Цепь представляется вектором хешей, где хеш — число размером 8 байт. Для расчета хешей используется хеш-функция `SipHash`. Для каждого очередного в качестве значения используются хеш-суммы всех файлов, имеющихся в куске, а также хеш-предыдущего куска (для первого куска используется 0).

Листинг 3: Расчет цепи хеш-сумм.

```
MergeTreeData::MergeTreeDataChainer::Checksums
MergeTreeData::MergeTreeDataChainer::calculateChain(
        const DataParts &data_parts) {
    Checksums result;
    result.reserve(data_parts.size());

    SipHash prev{0, 0};
    for (const auto &part : data_parts) {
        SipHash hash{0, 0};
        hash.update(prev);
        updateFromOnePart(hash, *part);

        const auto hash_value = hash.get64();
        prev = hash;
        result.push_back(hash_value);
    }
    return result;
}
void MergeTreeData::MergeTreeDataChainer::updateFromOnePart(
        SipHash &hash, const DataPart &data_part) {
    for (const auto &[file, checksum] : data_part.checksums.files) {
        hash.update(file);
        hash.update(checksum);
    }
}
```

Код метода `precommitChain` представлен в листинге 4. В данном методе происходят следующие действия:

1) проверятся консистентность цепи для текущего списка активных кусков;
2) если проверка успешна, то к текущему списку добавляется кусок на добавление, и удаляются куски, которые требуется удалить;
3) для нового состояние активных блоков высчитывается цепочка и записывается в журнал.

Листинг 4: Метод `precommitChain`.

```
MergeTreeData::MergeTreeDataChainer::CommitFinisherPtr
MergeTreeData::MergeTreeDataChainer::precommitChain(
    DataParts &data_parts, const DataPartPtr &part_to_add,
    const DataPartsVector &parts_to_remove, const DataPartsLock &lock)
{
    if (!checkConsistency(data_parts, lock).success) {
        if (log)
            log->error("Parts chain is inconsistent (force update: " +
                                    std::to_string(force_updates) + ")");
        if (!force_updates)
            return false;
    }
    transformToFutureState(data_parts, part_to_add, parts_to_remove);
    return precommitChain(data_parts, lock);
}

MergeTreeData::MergeTreeDataChainer::CommitFinisherPtr
MergeTreeData::MergeTreeDataChainer::precommitChain(
    const DataParts &data_parts, const DataPartsLock & /*lock*/)
{
    const auto checksum = calculateChain(data_parts);
    state.writePending({checksum});
    return true;
}

void MergeTreeData::MergeTreeDataChainer::transformToFutureState(
    DataParts &data_parts, const DataPartPtr &part_to_add,
    const DataPartsVector &parts_to_remove)
{
    data_parts.insert(part_to_add);
    for (const auto &part : parts_to_remove)
        data_parts.erase(part);
}
```

1.3 Код изменения состояний активных блоков

Как было описано в аналитической части и в схемах алгоритмов, улучшение метода хранения, которое позволяет доказывать неправомерное удаление, работает в тот момент, когда происходит изменения состояний блоков. Это происходит в функции `renameTempPartAndReplace` класса `MergeTreeData`. Код с использованием цепи хеш-сумм, описан в листингах 23 и 24 в приложении А.

2 Исследовательская часть

В данном разделе будет показаны действия, которые требуется сделать, чтобы обмануть систему без шифрования данных, а также будет показано, что система обнаруживает неправомерные действия с наличием шифрования.

2.1 Обход механизма защиты при отсутствии шифрования данных

Рассмотрение для 2 форматов хранения:

1) компактное;
2) широкое.

2.1.1 Компактное хранение

Необходимо создать таблицу и поместить в нее данные. Эти действия выполняются запросами, представленными в листинге 5.

Листинг 5: Создание таблицы с компактным хранением.

```
CREATE TABLE cats
(
    `id` UInt64 CODEC(NONE),
    `name` String CODEC(NONE),
    `age` UInt64 CODEC(NONE)
)
ENGINE = MergeTree
ORDER BY id
SETTINGS min_bytes_for_wide_part = 1000000000, min_rows_for_wide_part
   = 1000000000, use_parts_chainer = 1

INSERT INTO cats VALUES (1, 'Murka', 11), (2, 'Elsa', 18), (3, 'Cleo',
    5)
```

Как было показано в конструкторской части, при изменении какого-то блока данных требуется пересчитать 3 хеш-суммы куска и цепь хеш-сумм. Пусть требуется выставить столбцу значение $age = 4$, где $id = 3$. Данные на диске выглядят так, как показано в листинге 6.

Листинг 6: Данные записанного куска в сыром виде.

```
hexdump -C all_1_1_0/data.bin
39 32 97 f2 20 a1 0f 9e  d5 82 4c 8a 1c 64 a7 74  |92.. .....L..d.t|
02 21 00 00 00 18 00 00  00 01 00 00 00 00 00 00  |.!..............|
00 02 00 00 00 00 00 00  00 03 00 00 00 00 00 00  |................|
00 5c 43 37 3f 94 e6 65  eb 4d fb 60 68 3a 35 85  |.\C7?..e.M.`h:5.|
4d 02 19 00 00 00 10 00  00 00 05 4d 75 72 6b 61  |M..........Murka|
04 45 6c 73 61 04 43 6c  65 6f d8 cd aa 90 8a 0a  |.Elsa.Cleo......|
4a 0a 5f c3 97 cd bd e9  58 3d 02 21 00 00 00 18  |J._.....X=.!....|
00 00 00 0b 00 00 00 00  00 00 00 12 00 00 00 00  |................|
00 00 00 05 00 00 00 00  00 00 00                 |...........|
```

Если разобрать по байтам, получится:

1) 16 байт — хеш-сумма для сжатого блока (1 столбца);
2) 9 байт — заголовок кодека, состоящий из:
   — 1 байт — метод;
   — 4 байта — размер сжатых данных (с заголовком);
   — 4 байта — размер исходных данных.
3) 24 байта — 3 числа по 8 байт — столбец `id`;
4) 16 и 9 байт аналогично на хеш-сумму и заголовок кодека для 2 столбца;
5) 16 байт — строки «Murka», «Elsa» и «Cleo» и перед ними числа размером 1 байт;
6) 16 и 9 байт аналогично на хеш-сумму и заголовок кодека для 3 столбца;
7) 24 байта — 3 числа по 8 байт — столбец `age`.

Таким образом требуется поменять:

1) 1 байт — значение числа, которое хотим поменять;
2) 16 байт — хеш-сумма столбца (блока сжатых данных).

После замены файл `data.bin` будет выглядеть как в листинге 7.

Листинг 7: Данные записанного куска в сыром виде после исправления.

```
39 32 97 f2 20 a1 0f 9e  d5 82 4c 8a 1c 64 a7 74  |92.. .....L..d.t|
02 21 00 00 00 18 00 00  00 01 00 00 00 00 00 00  |.!..............|
00 02 00 00 00 00 00 00  00 03 00 00 00 00 00 00  |................|
00 5c 43 37 3f 94 e6 65  eb 4d fb 60 68 3a 35 85  |.\C7?..e.M.`h:5.|
4d 02 19 00 00 00 10 00  00 00 05 4d 75 72 6b 61  |M..........Murka|
04 45 6c 73 61 04 43 6c  65 6f 75 37 d2 0b 34 77  |.Elsa.Cleou7..4w|
72 89 c2 7a f3 1c 62 ef  ee 97 02 21 00 00 00 18  |r..z..b....!....|
00 00 00 0b 00 00 00 00  00 00 00 12 00 00 00 00  |................|
00 00 00 04 00 00 00 00  00 00 00                 |...........|
```

При проверке состояния таблицы получается результат, описанный в листинге 8.

Листинг 8: Результат проверки.

```
CHECK TABLE cats
format Vertical

Query id: dbcbc244-10f0-491d-85e7-849377ac73e0

Row 1:

part_path: all_1_1_0
is_passed: 0
message: Checksum mismatch for file data.bin in data part
```

Была заменена хеш-сумма сжатого блока, однако не были заменены хеш-суммы сжатых и расжатых данных, описанные в `checksums.txt`, который описан в листинге 9.

Листинг 9: Хеш-суммы, описанные в файле `checksums.txt`.

```
63 68 65 63 6b 73 75 6d  73 20 66 6f 72 6d 61 74  |checksums format|
20 76 65 72 73 69 6f 6e  3a 20 34 0a 67 cf 30 a7  | version: 4.g.0.|
25 ce b0 4e 0e 75 28 87  2a 91 be 42 82 8e 00 00  |%..N.u(.*..B....|
00 84 00 00 00 f1 3c 04  09 63 6f 75 6e 74 2e 74  |......<..count.t|
78 74 01 96 fc d9 4b 53  ab 22 6f 33 9a be c4 ce  |xt....KS."o3....|
28 da 5d 00 08 64 61 74  61 2e 62 69 6e 8b 01 5c  |(.]..data.bin..\|
70 1c a6 71 6a e6 e7 f3  d7 70 27 28 d3 99 12 01  |p..qj....p'(....|
40 24 e8 86 e7 dd 22 a9  de f5 3b 07 c6 0f 91 6d  |@$....".....;....m|
7e 09 2d 00 f0 25 6d 72  6b 33 70 eb db 30 3d 91  |~.-..%mrk3p..0=.|
74 86 b7 33 12 22 5a 25  89 00 cb 00 0b 70 72 69  |t..3."Z%.....pri|
6d 61 72 79 2e 69 64 78  10 f6 15 da b1 a5 50 1e  |mary.idx......P.|
18 54 ae 68 d7 22 30 aa  ee 00                    |.T.h."0...|
```

Хеш-суммы подсчитываются для файла с данными, файла с индексами, файла с количеством элементов в куске и файла с засечками.

Данные хранятся в сжатом виде, жмутся кодеком по умолчанию (LZ4). Порядок записи следующий:

— строка «checksums format version: 4»;
— хеш-сумма сжимающего буфера (16 байт);
— заголовок кодека (9 байт);
— количество файлов (как число переменной длины, в данном случае 1);
— для каждого файла:
  — имя файла;
  — размер файла;
  — хеш-сумма файла;
  — булевый признак сжатия;
  — размер расжатых данных (если булевый признак сжатия — истина);
  — хеш-сумма расжатых данных (если булевый признак сжатия — истина).

С учетом того, что менялся только `data.bin`, требуется поменять хеш-сумму только для него. Содержимое файла после изменений представлено в листинге 10. Хеш-сумма не была сжата, поэтому размер файла не изменился.

Листинг 10: Хеш-суммы, описанные в файле `checksums.txt` после внесения изменений.

```
 1 63 68 65 63 6b 73 75 6d  73 20 66 6f 72 6d 61 74  |checksums format|
 2 20 76 65 72 73 69 6f 6e  3a 20 34 0a 67 cf 30 a7  | version: 4.g.0.|
 3 25 ce b0 4e 0e 75 28 87  2a 91 be 42 82 8e 00 00  |%..N.u(.*..B....|
 4 00 84 00 00 00 f1 3c 04  09 63 6f 75 6e 74 2e 74  |......<..count.t|
 5 78 74 01 96 fc d9 4b 53  ab 22 6f 33 9a be c4 ce  |xt....KS."o3....|
 6 28 da 5d 00 08 64 61 74  61 2e 62 69 6e 8b 01 5c  |(.]..data.bin..\|
 7 70 1c a6 71 6a e6 e7 f3  d7 70 27 28 d3 99 12 01  |p..qj....p'(....|
 8 40 24 e8 86 e7 dd 22 a9  de f5 3b 07 c6 0f 91 6d  |@$....".....;....m|
 9 7e 09 2d 00 f0 25 6d 72  6b 33 70 eb db 30 3d 91  |~.-..%mrk3p..0=.|
10 74 86 b7 33 12 22 5a 25  89 00 cb 00 0b 70 72 69  |t..3."Z%.....pri|
11 6d 61 72 79 2e 69 64 78  10 f6 15 da b1 a5 50 1e  |mary.idx......P.|
12 18 54 ae 68 d7 22 30 aa  ee 00                    |.T.h."0...|
```

Вывод проверки после проведенных изменений показан в листинге 11.

Листинг 11: Результат проверки после исправления `checksums.txt`

```
CHECK TABLE cats
FORMAT Vertical

Query id: dbcbc244-10f0-491d-85e7-849377ac73e0

Row 1:

part_path: all_1_1_0
is_passed: 1
message:

Row 2:

part_path: parts chain
is_passed: 0
message:   One or more parts before all_1_1_0 were deleted OR part
   itself is not valid!
```

Проверка показывает, что теперь нарушена цепь хеш-сумм блоков. Полный алгоритм построения цепи хеш-сумм и код алгоритма описаны в конструкторском и технологическом разделах. Файл с цепью хеш-сумм содержит в себе информацию о массиве чисел размером 8 байт. для 1 куска содержится 16 байт информации. После пересчета хеш-суммы и замены значений в файле `commited_parts_chain.bin` вывод не поменяется, так как данные о значениях цепочки кэшируются в оперативной памяти. Данный факт вынуждает злоумышленника сделать так, чтобы СУБД перезапустилась. При перезапуске сервера значения цепи хеш-сумм поменяются на нужные и получится вывод, показанный в листинге 12.

Листинг 12: Результат проверки после исправления `commited_parts_chain.bin`.

```
CHECK TABLE cats
FORMAT Vertical

Query id: 95e1ae1c-203e-488b-a0a2-f3885ab4aff1

Row 1:

```

```
part_path: all_1_1_0
is_passed: 1
message:

Row 2:

part_path: parts chain
is_passed: 1
message:
```

Как итог, чтобы проверка куска проходила успешно, требуется:

1) пересчитать значение хеш-суммы сжатого блока, в который было внесено изменение;
2) пересчитать значение хеш-суммы файла `data.bin` в файле `checksums.txt`;
3) пересчитать значение цепи хеш-сумм в файле `commited_parts_chain.bin`.

При удалении потребуется сделать только последний шаг. Аналогично потребуется поступить при замене куска (то есть при удалении и вставке куска с аналогичной структурой). Подобные сценарии неправомерного доступа проходили бы при старом методе хранения данных.

2.1.2 Широкое хранение

Для широкого хранения следует рассмотреть отличия от приведенного выше алгоритма действий:

1) изменение сжатого блока и его хеш-суммы будет происходит в соответсвующем нужному столбцу файле;
2) изменение в файле `checksums.txt` будет происходить для файла, соответствующего меняемому столбцу.

2.2 Работа системы с шифрованием данных

Шифрование можно настроить на нескольких уровнях:

— на уровне отдельных столбцов;
— на уровне виртуального диска (пути в файловой системе, который будет передан ClickHouse через конфигурационный файл).

Шифрование на уровне столбцов позволяет злоумышленнику менять фай-

лы с индексами, а также файлы с засечками, что в совокупности позволяет сделать так, чтобы при выборке данных с условием на первичный ключ данные какого-либо куска не читались вовсе, что аналогично их удалению. Однако на запросы, которые не имеют ограничений на первичный ключ, данный подход не подействует.

Пусть имеется шифрование на уровне виртуального диска. Пример конфигурации для такого шифрования приведен в листинге 13.

Листинг 13: Параметры конфигурации для шифрования на уровне диска.

```
<storage_configuration>
    <disks>
        <disk1>
            <type>local</type>
            <path>/ClickHouse/build/programs/disk/</path>
        </disk1>
        <default>
            <type>encrypted</type>
            <disk>disk1</disk>
            <path>enc/</path>
            <algorithm>AES_256_CTR</algorithm>
            <key>1234000000000000123400000000000000</key>
        </default>
    </disks>
</storage_configuration>
```

Тогда в случае повторения запросов из листинга 5, данные, записанные в файл, будут выглядеть так, как показано в листинге 14.

Листинг 14: Данные при шифровании диска.

```
45 4e 43 01 00 02 00 00  00 00 00 00 00 00 00 05  |ENC.............|
31 8d 4f 11 c1 c1 3b 78  ae 20 b6 59 7b ff 67 f4  |1.O...;x. .Y{.g.|
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
f9 82 0a 75 5f 80 01 de  36 e4 22 e9 78 b1 5a 8e  |...u_...6.".x.Z.|
8a 24 e2 2b 2b 57 5b cc  71 18 96 ea af ce 21 e8  |.$.++W[.q.....!.|
e7 20 5c 8b 26 20 98 8d  e9 34 17 cb a9 82 72 43  |. \.& ...4....rC|
ba fc b6 3b f6 f2 78 a2  89 a0 a2 80 b5 4e e5 3b  |...;..x......N.;|
b3 30 f9 e5 6b 20 f5 c0  ef 77 a2 6b 2f ce 4e 0c  |.0..k ...w.k/.N.|
e5 31 af 8c 1c 2f 55 e1  27 9d db 70 24 bc 92 de  |.1.../U.'..p$...|
3e 8c 91 12 83 1c 90 1a  f7 2b a1 1a ce c4 56 c3  |>........+....V.|
a7 e0 62 f8 92 d7 c1 e5  23 eb 64 df 79 5f 15 3d  |..b.....#.d.y_.=|
02 b4 c0 e6 de 12 f0 b2  a4 f9 7e                 |..........~|
```

Шифрование не позволяет исправить хеш-суммы подобно тому, как это выполнялось без него. Требуется проверить:

- что работают все стандартные операции:
  - вставка;
  - слияние;
  - мутация.
- что неправомерные удаление и изменение блока обнаружаются.

2.2.1 Проверка работы стандартных операций

Для проверки корректности работы цепи хеш-сумм со вставкой можно сделать 3 вставки командой из листинга 5. Результат проверки таблицы в листинге 15.

Листинг 15: Результат проверки после вставки 4 блоков.

```
CHECK TABLE cats
FORMAT Vertical

Query id: 70296b3b-a0ec-4938-b731-a47310bc32b8

Row 1:

part_path: all_1_1_0
is_passed: 1
message:

Row 2:

part_path: all_2_2_0
is_passed: 1
message:

Row 3:

part_path: all_3_3_0
is_passed: 1
message:

Row 4:

part_path: all_4_4_0
is_passed: 1
message:

Row 5:

part_path: parts chain
is_passed: 1
message:
```

Для проверки слияния можно сделать еще 3 вставки. Результат проверки в листинге 16. Из проверки видно, что произошло слияние блоков с 1 по 5, цепь осталась валидной.

Листинг 16: Результат проверки после вставки 7 блоков.

```
CHECK TABLE cats
FORMAT Vertical

Query id: c0888295-5a25-4528-8a96-7e8ba7762d00

Row 1:

part_path: all_1_5_1
is_passed: 1
message:

Row 2:

part_path: all_6_6_0
is_passed: 1
message:

Row 3:

part_path: all_7_7_0
is_passed: 1
message:

Row 4:

part_path: parts chain
is_passed: 1
message:
```

Для проверки работы мутаций следует выполнить запрос из листинга 17. Результат проверки таблицы после проведения мутации представлен в листинге 18, цепь хеш-сумм осталась валидной.

Листинг 17: Запрос мутации.

```
ALTER TABLE cats UPDATE age=age + 1 WHERE age < 17;
```

Листинг 18: Проверка таблицы после мутации.

```
CHECK TABLE cats
FORMAT Vertical

Query id: aed93f24-7417-4424-8d16-cac981c1d240

Row 1:

part_path: all_1_5_1_8
is_passed: 1
message:

Row 2:

part_path: all_6_6_0_8
is_passed: 1
message:

Row 3:

part_path: all_7_7_0_8
is_passed: 1
message:

Row 4:

part_path: parts chain
is_passed: 1
message:
```

2.2.2 Проверка обнаружения неправомерных удаления и изменения кусков

Для проверки обнаружения неправомерного удаления можно удалить кусок `all_7_7_0_8`. Вывод проверки представлен в листинге 19, вывод проверки после перезагрузки системы представлен в листинге 20. В обоих случаях показано, что данные были изменены где-то вне системы.

Листинг 19: Вывод системы после неправомерного удаления куска.

```
CHECK TABLE cats
FORMAT Vertical

Query id: 6aa87471-30e7-4c8c-8c79-e3243ee936ae

Row 1:

part_path: all_1_5_1_8
is_passed: 1
message:

Row 2:

part_path: all_6_6_0_8
is_passed: 1
message:

Row 3:

part_path: all_7_7_0_8
is_passed: 0
message:   Check of part finished with error: 'Cannot open file /
   ClickHouse/build/programs/disk/enc/store/cf0/cf01ce53-cba4-4afb-8
   e11-26b487355f84/all_7_7_0_8/columns.txt, errno: 2, strerror: No
   such file or directory'
```

Листинг 20: Вывод системы после неправомерного удаления куска после перезагрузки системы.

```
CHECK TABLE cats
FORMAT Vertical

Query id: 362f64b5-a480-4f2a-b1ad-0c021a110c53

Row 1:

part_path: all_1_5_1_8
is_passed: 1
message:

Row 2:

part_path: all_6_6_0_8
is_passed: 1
message:

Row 3:

part_path: parts chain
is_passed: 0
message:   Some parts in the end of the chain were deleted
```

Для проверки обнаружения неправомерного изменения можно изменить последовательность байт в зашифрованном файле с данными одного из кусков, например, `all_7_7_0_8`. После изменения данных командой из листинга 21 получается результат проверки из листинга 22. Данное было бы обнаружено и до оптимизации метода, потому что, как было замечено в констуркторской части, при изменении данных изменяются еще и хеш-суммы файлов и сжатых блоков, которые были и до модификации метода.

Листинг 21: Неправомерное изменение данных.

```
printf '\x31' | dd of=cats/all_7_7_0_8/data.bin bs=1 seek=183 count=1
   conv=notrunc
```

Листинг 22: Результат проверки после неправомерного изменения данных.

```
CHECK TABLE cats
FORMAT Vertical

Query id: 3b86fbe1-b13e-4006-8e92-0d460a52784f

Row 1:

part_path: all_1_5_1_8
is_passed: 1
message:

Row 2:

part_path: all_6_6_0_8
is_passed: 1
message:

Row 3:

part_path: all_7_7_0_8
is_passed: 0
message:   Checksum mismatch for file data.bin in data part
```

Вывод

В данном разделе были рассмотрены способы обхода защиты системы, работающей без шифрования, было показано правильное функционирование с шифрованием и применением стандартных операций, а также проверена возможность обнаружения неправомерных действий.

ЗАКЛЮЧЕНИЕ

В рамках данной работы:

— были описаны инструменты и технологии, требуемые для реализации нового метода;

— был реализован новый метод в виде класса, отвечающего за управление цепью хеш-сумм, связывающую куски данных;

— полученная реализация класса была встроена в функционал движка MergeTree;

— метод был исследован на предмет возможности реализации угроз (обхода защиты) при нешифрованном хранении данных;

— была исследована корректность работы метода при различных событиях системы при шифрованном хранении данных.

Таким образом цель работы — реализовать улучшение метода блочного хранения данных в СУБД ClickHouse в движке MergeTree, дающее возможность доказательства неправомерного доступа была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Руководство по языку C++. [Электронный ресурс]. – Режим доступа: https://en.cppreference.com/w/ свободный – (21.03.2022).

2. Выбор компилятора для сборки ClickHouse. [Электронный ресурс]. – Режим доступа: https://clickhouse.com/docs/en/development/developer-instruction/c-compiler свободный – (24.03.2022).

3. Официальный сайт Clang. [Электронный ресурс]. – Режим доступа: https://clang.llvm.org/ свободный – (24.03.2022).

4. Официальный сайт системы сборки CMake. [Электронный ресурс]. – Режим доступа: https://cmake.org/ свободный – (27.03.2022).

5. Официальный сайт системы сборки Ninja. [Электронный ресурс]. – Режим доступа: https://ninja-build.org/ свободный – (27.03.2022).

ПРИЛОЖЕНИЕ А

Листинг 23: Изменение состояний блоков (переименование блока).

```
bool MergeTreeData::renameTempPartAndReplace(
    MutableDataPartPtr &part, MergeTreeTransaction *txn,
    SimpleIncrement *increment, Transaction *out_transaction,
    std::unique_lock<std::mutex> &lock, DataPartsVector *
   out_covered_parts,
    MergeTreeDeduplicationLog *deduplication_log,
    std::string_view deduplication_token)
{
    // ...
    part->name = part_name;
    part->info = part_info;
    part->is_temp = false;
    part->setState(DataPartState::PreActive);

    if (parts_chainer) {
        DataParts active_parts;
        auto range = getDataPartsStateRange(DataPartState::Active
   );
        active_parts.insert(range.begin(), range.end());
        auto need_commit_chain =
                parts_chainer->precommitChain(active_parts, part,
    covered_parts, lock);

        part->renameTo(part_name, true);

        if (need_commit_chain)
            parts_chainer->commitChain(lock);
    } else {
        part->renameTo(part_name, true);
    }

    auto part_it = data_parts_indexes.insert(part).first;
    MergeTreeTransaction::addNewPartAndRemoveCovered(
   shared_from_this(), part,
        covered_parts, txn);
```

Листинг 24: Изменение состояний блоков (переименование блока). Продолжение.

```
    if (out_transaction) {
        out_transaction->precommitted_parts.insert(part);
    } else {
        size_t reduce_bytes = 0;
        size_t reduce_rows = 0;
        size_t reduce_parts = 0;
        auto current_time = time(nullptr);
        for (const DataPartPtr &covered_part : covered_parts) {
            covered_part->remove_time.store(current_time, std::
    memory_order_relaxed);
            modifyPartState(covered_part, DataPartState::Outdated
    );
            removePartContributionToColumnAndSecondaryIndexSizes(
    covered_part);
            reduce_bytes += covered_part->getBytesOnDisk();
            reduce_rows += covered_part->rows_count;
            ++reduce_parts;
        }

        modifyPartState(part_it, DataPartState::Active);
        addPartContributionToColumnAndSecondaryIndexSizes(part);
        // ...
    }
}
```