???? ??????

CIS Benchmarks (CIS - Center for Internet Security) — это набор рекомендаций и стандартов безопасности информационных систем, разработанный организацией Center for Internet Security. Каждый мануал представляет собой набор наилучших практик и конфигураций для различных операционных систем, устройств и ПО.

Рекомендации по настройке могут различаться в зависимости от типа базы данных (например, PostgreSQL, MySQL, Microsoft SQL Server и т. д.) и версии базы данных. Но общие принципы и рекомендации по настройке обязательно включают в себя:

- Аутентификация и авторизация: права доступа к базе данных должны быть ограничены для каждого пользователя на минимально необходимый уровень, а учетные записи пользователей должны иметь сильные пароли, также по возможности стоит использовать двухфакторную аутентификацию.
- Шифрование: в большинстве СУБД (систем управления базами данных) существуют механизмы встроенного шифрования, которые следует активировать. Помимо этого, необходимо шифровать соединений с базой данных с помощью SSL/TLS.
- Аудит и мониторинг: в базах данных должны быть установлены механизмы мониторинга для отслеживания активности пользователей и обнаружения подозрительных действий, например: создание учетной записи с широкими правами, выполнение очень большого запроса, и т.д. Также рекомендуется проводить регулярный аудит учетных записей на предмет превышения прав.
- Обновления и патчи: регулярно обновляйте базу данных и устанавливайте патчи для закрытия известных уязвимостей.
- Защита от инъекций: применяйте методы предотвращения инъекций, такие как параметризованные запросы и санитаризация входных данных, чтобы предотвратить SQL-инъекции и другие виды атак.
- Реагирование на инциденты безопасности: разработайте план реагирования на инциденты и обучите свой персонал его исполнению в случае возникновения инцидентов.

?????????? ? ?????????? ???????? ? ????? ??????

Большинство современных СУБД предоставляют возможность вести журнал запросов, записывая все запросы, поступающие к базе данных. Помимо такого "внутреннего" логирования существуют системы мониторинга баз данных. Они могут непрерывно отслеживать активность баз данных, включая выполняемые запросы, количество обращений

и нагрузку на систему. Примеры таких систем — Dynatrace, Datadog.

В данном курсе мы рассмотрим прежде всего возможности логирования внутри баз данных. На основании этих логов можно самостоятельно создавать правила SIEM для выявления несанкционированных или подозрительных операций.

?????????? ???????? ? PostgreSQL

Для мониторинга запросов в базах данных PostgreSQL можно использовать различные инструменты и методы. Ниже представлено несколько способов:

1. **Журналы PostgreSQL:** PostgreSQL записывает информацию о выполненных запросах в журналы ошибок (`log_error_verbosity`) и журналы запросов (`log_statement`). Вы можете настроить эти параметры в конфигурационном файле PostgreSQL (например, *postgresql.conf*) и перезапустить сервер:

   ```
   log_statement = 'all' log_destination = 'stderr' logging_collector = on
   ```
   

   После настройки PostgreSQL будет записывать все SQL запросы в указанный в log_destination журнал.
2. **Утилита pg_stat_statements:** это встроенный модуль PostgreSQL, который отслеживает статистику выполнения SQL запросов. Вы можете включить его в конфигурационном файле PostgreSQL и перезапустить сервер:

   ```
   shared_preload_libraries = 'pg_stat_statements' pg_stat_statements.max = 10000
   pg_stat_statements.track = all
   ```
   

   Затем можно выполнить запросы к представлению `pg_stat_statements` для получения информации о наиболее часто выполняемых запросах, их времени выполнения и других метриках.

Пример SQL запроса для просмотра наиболее часто выполняемых запросов и их времени выполнения с использованием `pg_stat_statements`:

```
SELECT query, total_time, calls, rows FROM pg_stat_statements ORDER BY total_time DESC LIMIT
10;
```


Этот запрос покажет 10 наиболее ресурсоемких запросов по времени выполнения, их количество вызовов и количество обработанных строк.

????????? ???????????? ? PostgreSQL

Настройки логирования возможно изменить в файле `postgresql.conf`, который обычно располагается в каталоге `postgresql`.

Как это сделать:

1. **Настройте параметры логирования**:
   В файле `postgresql.conf` найдите и отредактируйте следующие параметры:
   - `log_statement = 'all'`: параметр определяет, какие SQL-запросы будут записываться в журнал. Установите его значение на `all`, чтобы записывать все SQL-запросы.
   - `log_connections = on`: параметр указывает, нужно ли записывать информацию о подключениях к базе данных в журнал. Установите его значение на `on`, чтобы записывать информацию о подключениях.
   - `log_disconnections = on`: параметр указывает, нужно ли записывать информацию о отключениях от базы данных в журнал. Установите его значение на `on`, чтобы записывать информацию об отключениях.
   - `log_duration = on`: параметр указывает, нужно ли записывать информацию о продолжительности выполнения SQL-запросов в журнал. Установите его значение на `on`, чтобы записывать информацию о продолжительности выполнения запросов.
2. **Перезапустите PostgreSQL**:
   После внесения изменений в файл `postgresql.conf` перезапустите PostgreSQL для применения новых настроек:
   `sudo service postgresql restart`

После выполнения этих шагов PostgreSQL будет записывать административные действия, такие как создание ролей и назначение прав доступа, в файл журнала `postgresql.log` с необходимой детализацией, в соответствии с указанными параметрами.

Пример файла postgresql.conf

```
#------------------------------------------------------------------------------
# FILE LOCATIONS
#------------------------------------------------------------------------------

data_directory = '/var/lib/postgresql/12/main'

#------------------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------------
```

```
listen_addresses = 'localhost'
port = 5432

#------------------------------------------------------------------------------
# RESOURCE USAGE (except WAL)
#------------------------------------------------------------------------------

shared_buffers = 128MB

#------------------------------------------------------------------------------
# WRITE AHEAD LOG
#------------------------------------------------------------------------------

wal_level = replica
archive_mode = on
archive_command = 'cp %p /var/lib/postgresql/12/main/archive/%f'

#------------------------------------------------------------------------------
# QUERY TUNING
#------------------------------------------------------------------------------

max_connections = 100
effective_cache_size = 4GB
work_mem = 64MB

#------------------------------------------------------------------------------
# ERROR REPORTING AND LOGGING
#------------------------------------------------------------------------------

log_destination = 'stderr'
logging_collector = on
log_directory = 'pg_log'
log_filename = 'postgresql-%Y-%m-%d.log'
log_statement = 'all'
log_connections = on
log_disconnections = on
log_duration = on
```

```
#------------------------------------------------------------------------------
# SECURITY AND AUTHORIZATION
#------------------------------------------------------------------------------

password_encryption = 'scram-sha-256'

#------------------------------------------------------------------------------
# OTHERS
#------------------------------------------------------------------------------

max_wal_size = 2GB
```

???? ??????? ? ??????? postgresql.log

Сообщения сервера — информационные сообщения о состоянии сервера, его настройках, запущенных процессах и т. д.:

```
2024-03-28 10:00:00.123 UTC [23345] LOG:  database system is ready to accept connections
```

Сообщения об ошибках, возникающих во время выполнения запросов, операций и т. д.:

```
2024-03-28 10:05:00.456 UTC [23345] ERROR:  syntax error at or near "SELECT"
```

Записи о выполняемых SQL-запросах, включая их текст, время выполнения, идентификаторы процессов и т. д.:

```
2024-03-28 10:10:00.789 UTC [23345] LOG:  statement: SELECT * FROM users;
2024-03-28 10:00:01.234 UTC [23345] LOG: statement: INSERT INTO products (name, price) VALUES ('Product A', 99.99);
2024-03-28 10:00:02.345 UTC [23345] LOG: statement: UPDATE customers SET email = 'new@example.com' WHERE id
= 123;
```

Сообщения о подключениях и отключениях к серверу PostgreSQL:

```
2024-03-28 10:15:00.012 UTC [23345] LOG:  connection received: host=127.0.0.1 port=5432
2024-03-28 10:20:00.345 UTC [23345] LOG:  disconnection: session time: 0:05:00
```

Информация о процессах репликации, восстановлении, синхронизации и т. д., если сервер работает в режиме репликации.

```
2024-03-28 10:25:00.678 UTC [23345] LOG:  starting streaming replication
```

Сообщения о сбоях и восстановлении:

```
2024-03-28 10:30:00.901 UTC [23345] LOG:  server process (PID 23345) was terminated by signal 9: Killed
2024-03-28 10:35:00.234 UTC [23345] LOG:  database system was interrupted; last known up at 2024-03-28 10:00:00 UTC
```

Сообщения о процессах архивации и восстановления:

```
2024-03-28 10:40:00.567 UTC [23345] LOG:  starting online backup of database "mydb"
2024-03-28 10:45:00.890 UTC [23345] LOG:  restore of database "mydb" from archive completed
```

Сообщения о настройках и изменениях конфигурации:

```
2024-03-28 10:50:00.123 UTC [23345] LOG:  configuration file "/etc/postgresql.conf" contains changes
```

Рассмотрим подробнее пример файла журнала `postgresql.log` с включенной опцией `log_statement`, который записывает в лог все SQL-запросы:

```
2024-03-28 10:00:00.123 UTC [23345] LOG: statement: SELECT * FROM users;
2024-03-28 10:00:01.234 UTC [23345] LOG: statement: INSERT INTO products (name, price) VALUES ('Product A', 99.99);
2024-03-28 10:00:02.345 UTC [23345] LOG: statement: UPDATE customers SET email = 'new@example.com' WHERE id = 123;
```

В этом примере каждая строка в файле журнала представляет собой SQL-запрос, который был выполнен в базе данных. Каждая запись начинается с даты и времени события, за которыми следует идентификатор процесса PostgreSQL (`23345` в данном случае), уровень журналирования (`LOG`), и непосредственно сам SQL-запрос.

- **Дата и время (`2024-03-28 10:00:00.123 UTC`):** метка времени, когда был выполнен SQL-запрос.
- **Идентификатор процесса (`[23345]`):** это уникальный идентификатор процесса PostgreSQL, который выполнял SQL-запрос.
- **Уровень журналирования (`LOG`):** указывает на уровень важности сообщения. В данном случае, это просто информационное сообщение.
- **SQL-запрос (`statement: ...`):** это сам SQL-запрос, который был выполнен. В примере показаны три различных SQL-запроса: выборка (`SELECT`), вставка (`INSERT`) и обновление (`UPDATE`).

?????????? ???????? ? MongoDB

1. **Профилирование MongoDB:** MongoDB позволяет включить профилирование, чтобы записывать информацию о выполненных запросах. Это делается с помощью команды `db.setProfilingLevel()`.

Пример установки уровня профилирования в MongoDB:

```
db.setProfilingLevel(1, { slowms: 100 });
```

Значение **1** указывает на включение профилирования. Это означает, что MongoDB будет записывать информацию о каждом выполненном запросе.
{ slowms: 100 }: это опциональный параметр, который определяет пороговое значение времени выполнения запроса в миллисекундах, после которого запрос считается "медленным". В данном случае запросы, выполняющиеся дольше 100 миллисекунд, будут считаться медленными и будут записываться в профилировочный журнал.

2. **Использование инструментов мониторинга:** для мониторинга MongoDB можно использовать различные инструменты, такие как MongoDB Compass, MongoDB Cloud Manager, MongoDB Ops Manager и другие. Они предоставляют дашборды и отчеты о выполненных запросах и производительности сервера MongoDB.
3. **db.currentOp():** этот метод позволяет просматривать текущие операции в MongoDB, включая выполняемые запросы.

???????? ??????? ???????? ? MongoDB

В MongoDB метод `db.currentOp()` используется для получения списка текущих операций, выполняющихся в базе данных. Этот метод полезен для мониторинга текущей активности и идентификации долго выполняющихся операций или блокировок. Пример вывода после использования `db.currentOp()`:

Раскрыть

```
{
   "inprog" : [
      {
         "opid" : 6146937,
         "active" : true,
         "secs_running" : 5,
         "op" : "query",
         "ns" : "test.collection",
         "command" : {
            "find" : "collection",
            "filter" : {
               "status" : "active"
            },
```

```
      "lsid" : {
         "id" : UUID("50cb6696-739b-4db3-806d-31a4365a4e38")
      },
      "$clusterTime" : {
         "clusterTime" : Timestamp(1632785477, 1),
         "signature" : {
            "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
            "keyId" : NumberLong(0)
         }
      },
      "originatingCommand" : {
         "find" : "collection",
         "filter" : {
            "status" : "active"
         },
         "lsid" : {
            "id" : UUID("50cb6696-739b-4db3-806d-31a4365a4e38")
         },
         "$clusterTime" : {
            "clusterTime" : Timestamp(1632785477, 1),
            "signature" : {
               "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
               "keyId" : NumberLong(0)
            }
         },
         "...
   },
   "planSummary" : "COLLSCAN",
   "numYields" : 0,
   "locks" : {
      "ReplicationStateTransition" : {
         "acquireCount" : {
            "w" : NumberLong(1)
         }
      },
      "Global" : {
         "acquireCount" : {
            "r" : NumberLong(1)
         }
```

```
      },
      "Database" : {
         "acquireCount" : {
            "r" : NumberLong(1)
         }
      },
      "Collection" : {
         "acquireCount" : {
            "r" : NumberLong(1)
         }
      }
   },
   "responseLength" : 15825718,
   "protocol" : "op_msg",
   "millis" : 5,
   "execStats" : {

   },
   "ts" : ISODate("2021-09-28T14:11:17.631Z"),
   "client" : "127.0.0.1",
   "appName" : "MongoDB Shell",
   "clientMetadata" : {
      "application" : {
         "name" : "MongoDB Shell"
      },
      "driver" : {
         "name" : "MongoDB Internal Client",
         "version" : "4.4.8"
      },
      "os" : {
         "type" : "Linux",
         "name" : "Ubuntu",
         "architecture" : "x86_64",
         "version" : "20.04"
      }
   },
   "numYields" : 0,
   "locks" : {
```

```
         },
         "waitingForLock" : false,
         "awaiting_historical_lock" : false,
         "lockStats" : {
            "ReplicationStateTransition" : {
               "timeLockedMicros" : {
                  "r" : NumberLong(0),
                  "w" : NumberLong(87)
               }
            },
            "Global" : {
               "timeLockedMicros" : {
                  "r" : NumberLong(0),
                  "w" : NumberLong(87)
               }
            },
            "Database" : {
               "timeLockedMicros" : {
                  "r" : NumberLong(0),
                  "w" : NumberLong(87)
               }
            },
            "Collection" : {
               "timeLockedMicros" : {
                  "r" : NumberLong(0),
                  "w" : NumberLong(87)
               }
            }
         }
      }
   ],
   "ok" : 1
}
```

В примере приведена информация о текущей операции, выполняющейся в MongoDB. Поля в этом логе:

1. **opid**: Уникальный идентификатор операции (Operation ID).

2. **active**: Показывает, активна ли операция в данный момент.
3. **secs_running**: Время, в течение которого операция выполняется, в секундах.
4. **op**: Тип операции. В данном случае это `query`, что означает выполнение запроса к базе данных.
5. **ns**: Пространство имен, в котором выполняется операция. В данном случае запрос выполняется в коллекции `test.collection`.
6. **command**: Команда запроса. Здесь показана конкретная команда `find` для поиска документов в коллекции `collection`, которые соответствуют фильтру `{ "status" : "active" }`.
7. **planSummary**: Краткое описание плана выполнения запроса. В данном случае используется коллекционный скан (`COLLSCAN`), что может указывать на то, что индекс не используется для выполнения запроса.
8. **locks**: Информация о блокировках, которые удерживает операция. Здесь показаны различные типы блокировок, включая блокировки на уровне глобального объекта, базы данных и коллекции.
9. **responseLength**: Длина ответа на запрос, в байтах.
10. **client**: IP-адрес или хост, откуда был отправлен запрос.
11. **appName**: Имя приложения, которое инициировало запрос. В данном случае это MongoDB Shell.
12. **clientMetadata**: Дополнительная метаданные о клиенте, включая информацию о драйвере и операционной системе.
13. **waitingForLock**: Показывает, ожидает ли операция блокировку.
14. **awaiting_historical_lock**: Показывает, ожидает ли операция историческую блокировку.
15. **lockStats**: Статистика блокировок для операции, включая время, в течение которого различные типы блокировок были удерживаемыми.

????????? ???????????? ? MongoDB

В MongoDB настройка логирования осуществляется через файл конфигурации `mongod.conf`.

1. **Настройка параметров логирования**:
   В файле `mongod.conf` найдите и отредактируйте параметры, связанные с логированием:
   - `systemLog.destination`: Укажите тип журнала, который вы хотите использовать. Например, `file` для записи в файл или `syslog` для записи в системный журнал.
   - `systemLog.path`: Укажите путь к файлу журнала, если вы выбрали тип `file`.
   - `systemLog.logAppend`: Если установлено в `true`, новые записи будут добавляться в конец существующего файла журнала. Если установлено в `false`, файл будет перезаписываться при каждом запуске.
   - `systemLog.verbosity`: Укажите уровень журналирования. Например, `0` для минимального уровня (только критические сообщения), `1` для информационных сообщений, `2` для отладочных сообщений и т. д.
2. **Перезапустите MongoDB**:

После внесения изменений в файл `mongod.conf` перезапустите MongoDB для применения новых настроек.
`sudo service mongod restart`

Пример файла mongod.conf

```
# mongod.conf

# Установка каталога данных и журнала
storage:
  dbPath: /var/lib/mongodb
  journal:
    enabled: true

# Установка параметров системного журнала
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log
  verbosity: 1

# Установка параметров сети
net:
  port: 27017
  bindIp: 127.0.0.1

# Настройка параметров безопасности
security:
  authorization: enabled

# Другие параметры конфигурации (необязательно)
```

???? MongoDB

В логе mongod.log MongoDB записывает различные типы событий, связанных с работой сервера и его окружения.

Основные типы событий, которые могут присутствовать в журнале mongod.log:

Сообщения о запуске и остановке сервера:

```
2024-03-28T12:00:00.000+0000 I CONTROL  [main] ***** SERVER RESTARTED *****
2024-03-28T12:00:05.000+0000 I CONTROL  [initandlisten] MongoDB starting : pid=123 port=27017 ...
```

- Сообщения об успешном запуске сервера MongoDB.
- Сообщения о завершении работы сервера при выключении.

Сообщения о подключении и отключении клиентов:

```
2024-03-28T12:05:23.000+0000 I NETWORK  [listener] connection accepted from 192.168.1.100:54321 #1 (1 connection now open)
2024-03-28T12:05:30.000+0000 I NETWORK  [conn1] end connection 192.168.1.100:54321 (0 connections now open)
```

- Информация о новых клиентских подключениях к серверу MongoDB.
- Уведомления об отключении клиентов от сервера.

Сообщения об аутентификации и авторизации:

```
2024-03-28T12:10:12.000+0000 I ACCESS   [conn123] Successfully authenticated as user1
2024-03-28T12:10:20.000+0000 I ACCESS   [conn123] Authentication failed for user2 from IP 192.168.1.101
```

- Логирование попыток аутентификации пользователей и ролей.
- Уведомления о результате аутентификации (успех или неудача).

Сообщения о выполнении операций:

```
2024-03-28T12:15:45.000+0000 I COMMAND  [conn123] command mydatabase.collection command: find { find: "collection", filter: {}, ...
2024-03-28T12:15:50.000+0000 I WRITE    [conn123] update mydatabase.collection query: { _id: ObjectId("123") } update: ...
```

- Логирование различных операций с данными, таких как запросы на чтение и запись, обновление и удаление документов.
- Информация о долго выполняющихся запросах и запросах, требующих индексов.

Сообщения об ошибках и предупреждениях:

```
2024-03-28T12:20:00.000+0000 W STORAGE  [conn123] Detected unclean shutdown - /data/db/mongod.lock is not empty.
2024-03-28T12:20:10.000+0000 E STORAGE  [conn123] WiredTiger error (0) [111111111] at <path/to/file>:123, terminating
```

- Логирование различных видов ошибок, возникающих во время работы сервера MongoDB.
- Предупреждения о потенциальных проблемах и нежелательных событиях.

Сообщения о репликации и кластеризации:

```
2024-03-28T12:25:00.000+0000 I REPL     [rsSync] Starting initial sync with primary
2024-03-28T12:25:10.000+0000 I REPL     [rsSync] Initial sync attempt failed, retrying in 10 seconds
```

- Информация о состоянии репликации и синхронизации данных между узлами кластера.
- Логирование операций репликации, таких как выборы лидера, перенос данных и т. д.

Сообщения о резервном копировании и восстановлении:

```
2024-03-28T12:30:00.000+0000 I COMMAND  [conn123] command admin.backup command: backup { backup: "mydatabase" }
2024-03-28T12:30:05.000+0000 I COMMAND  [conn123] command admin.restore command: restore { restore: "mydatabase" }
```

- Информация о процессах резервного копирования и восстановления данных.
- Логирование успешных и неудачных операций резервного копирования и восстановления.

Сообщения о настройках и изменениях конфигурации:

```
2024-03-28T12:35:00.000+0000 I CONTROL  [main] Setting featureCompatibilityVersion to "4.0"
2024-03-28T12:35:10.000+0000 I STORAGE  [initandlisten] Detected data files in /data/db created by the 'wiredTiger' storage engine, ...
```

- Уведомления о изменениях в конфигурационных файлах сервера MongoDB.
- Информация о перезагрузке сервера в связи с изменениями в конфигурации.

???? ? ????? ??????????????

? ??????????? ????? ??????

???? ????????????

Суперпользователь (Superuser) — особая роль, которая обладает полными правами на управление базой данных и ее объектами. Суперпользователь может выполнять любые операции с данными, включая создание, изменение и удаление объектов базы данных, а также управление пользователями и их правами.

Пользователь (User) — обычный пользователь базы данных, который имеет ограниченные права на выполнение операций с данными. Пользователи могут иметь доступ только к определенным объектам базы данных и выполнять только определенные операции, в зависимости от их назначенных прав доступа.

Роли безопасности (Security Roles) — являются наборами прав доступа, которые можно назначить одному или нескольким пользователям. Использование ролей позволяет упростить управление правами доступа и обеспечить согласованность безопасности в системе.

????? ???????

Права на объекты — определяют, какие операции разрешены для выполнения с определенными объектами базы данных, такими как таблицы, представления, процедуры и т.д. Типичные права включают SELECT (чтение), INSERT (добавление), UPDATE (обновление), DELETE (удаление) и другие.

Глобальные права — определяют операции, разрешенные для выполнения на уровне базы данных или даже на уровне сервера баз данных в целом. Например, глобальные права могут включать права на создание баз данных, создание пользователей, управление ролями и т.д.

Права на системные объекты — определяют доступ к системным объектам базы данных, таким как системные таблицы и представления, а также каталоги базы данных.

Права на схемы (Schema Privileges) — некоторые СУБД предоставляют возможность управления правами доступа на уровне схем, что позволяет более гибко управлять доступом к группам объектов.

??????? ???????? ????????????? ? PostgreSQL

В PostgreSQL для создания пользователя используется команда `CREATE ROLE`. Создавать новых пользователей могут пользователями с такими правами:

- **CREATE ROLE**: эта привилегия дает пользователю возможность создавать новые роли (включая пользователей) в базе данных PostgreSQL. Пользователи, обладающие этой привилегией, могут выполнять команду CREATE ROLE для создания новых пользователей.
- **SUPERUSER**: пользователь с правами суперпользователя (SUPERUSER) имеет все привилегии в базе данных, включая возможность создавать новых пользователей и назначать им любые права доступа.

Рассмотрим примеры создания пользователей с различными уровнями привилегий:

???????? ????????????????

```
CREATE ROLE superuser LOGIN SUPERUSER PASSWORD 'password';
```

Этот запрос создает нового суперпользователя с именем superuser и устанавливает пароль password. Суперпользователь имеет полный доступ ко всем объектам базы данных и может выполнять любые операции.

Как это будет выглядеть в логе:

```
<timestamp> LOG:  new superuser superuser created
<timestamp> LOG:  granting SUPERUSER privileges to user "superuser"
```

???????? ???????????? ? ????????????? ????????????

```
CREATE ROLE limited_user LOGIN PASSWORD 'password' VALID UNTIL '2025-12-31';
GRANT SELECT ON ALL TABLES IN SCHEMA public TO limited_user;
```

Этот запрос создает нового пользователя с именем limited_user, устанавливает ему пароль password и устанавливает срок его действия до 31 декабря 2025 года. Затем с помощью команды `GRANT` предоставляются ограниченные привилегии на чтение (SELECT) для всех таблиц в схеме public.

Как это будет выглядеть в логе:

```
<timestamp> LOG:  new user limited_user with password created
<timestamp> LOG:  granting LOGIN privileges to user "limited_user"
<timestamp> LOG:  granting SELECT privileges on all tables in schema public to user "limited_user"
```

???????? ???? ? ???????????????? ????????????

```
CREATE ROLE admin_role;
GRANT CREATE ROLE TO admin_role;
GRANT CREATE DATABASE TO admin_role;
```

Этот запрос создает новую роль admin_role, которая имеет право создавать другие роли и базы данных. Роль с административными привилегиями может быть использована для управления пользователями и базами данных в системе.

Как это будет выглядеть в логе:

```
<timestamp> LOG:  new role admin_role created
<timestamp> LOG:  granting CREATE ROLE privileges to role "admin_role"
<timestamp> LOG:  granting CREATE DATABASE privileges to role "admin_role"
```

????? ??????? ???????

Относительно SQL-команд повышенные права должны иметь только суперпользователи. Обычные пользователи PostgreSQL не должны обладать возможностью создавать роли, создавать новые базы данных, управлять репликацией или выполнять любое другое действие, считающееся привилегированным. Обычным пользователям должен предоставляться только минимальный набор привилегий, соответствующий управлению приложением:

- DDL (Data definition language - создание таблиц, создание представлений, создание индексов и т. д.);
- DML (Data Manipulation Language - select, insert, update, delete).

Также считается хорошей практикой создавать отдельные роли для DDL и DML. Для приложения с названием "payroll" были бы созданы следующие пользователи:

- payroll_owner;
- payroll_user.

Любые привилегии DDL предоставляются только учетной записи payroll_owner, в то время как привилегии DML предоставляются только учетной записи payroll_user. Это предотвращает случайное создание/изменение/удаление объектов базы данных кодом приложения, который выполняется от имени учетной записи payroll_user.

Таким образом, не ограничивая глобальные административные команды только суперпользователями, обычные пользователи, которым предоставлены избыточные привилегии, могут выполнять административные команды с неожиданными и нежелательными результатами.

В процессе аудита сначала необходимо проверить привилегии, предоставленные суперпользователю базы данных (определенному здесь как postgres), используя команду отображения `psql -c "\du postgres"`, чтобы установить базовую линию для предоставленных административных привилегий. На основе примера ниже, суперпользователь postgres может создавать роли, создавать базы данных, управлять репликацией и обходить уровни безопасности строк (RLS - Row Level Security):

```
# whoami
postgres
# psql -c "\du postgres"
                                   List of roles
Role name |                         Attributes                         | Member of
----------+------------------------------------------------------------+-----------
postgres  | Superuser, Create role, Create DB, Replication,            | {}
          |                         Bypass RLS                         |
```

Теперь проверим ту же информацию для обычного пользователя с именем appuser с помощью команды отображения `psql -c "\du appuser"`. Вывод подтверждает, что обычный пользователь appuser имеет те же повышенные привилегии, что и системный администратор пользователь postgres, так быть не должно:

```
# whoami
postgres
# psql -c "\du appuser"
                                   List of roles
Role name |                         Attributes                         | Member of
----------+------------------------------------------------------------+-----------
appuser   | Superuser, Create role, Create DB, Replication,  |        {}
          |                         Bypass RLS                         |
```

Этот пример показывает, что одному пользователю были присвоены излишние административные права, однако в процессе аудита нужно провести полный анализ всех пользователей базы данных, чтобы убедиться, что у них нет избыточных прав. Это можно сделать с помощью следующих команд:

```
# whoami
postgres
# psql -c "\du *"
# psql -c "select * from pg_user order by usename"
```

Если каким-либо обычным пользователям были предоставлены избыточные права, их следует немедленно отозвать с помощью команды SQL `ALTER ROLE`. Используя тот же пример выше, следующие команды отзывают все ненужные повышенные права у обычного пользователя appuser:

```
# whoami
postgres
# psql -c "ALTER ROLE appuser NOSUPERUSER;"
ALTER ROLE
# psql -c "ALTER ROLE appuser NOCREATEROLE;"
ALTER ROLE
# psql -c "ALTER ROLE appuser NOCREATEDB;"
ALTER ROLE
# psql -c "ALTER ROLE appuser NOREPLICATION;"
ALTER ROLE
# psql -c "ALTER ROLE appuser NOBYPASSRLS;"
ALTER ROLE
# psql -c "ALTER ROLE appuser NOINHERIT;"
ALTER ROLE
```

После выполненных действий нужно проверить новые права у пользователя appuser:

```
# whoami
postgres
# psql -c "\du appuser"
                  List of roles
Role name | Attributes | Member of
----------+------------+-----------
appuser   |            | {}
```

???? ? ????? ????????????? ? NoSQL ?????

В NoSQL базах данных, в отличие от реляционных, роли и права пользователей могут быть менее формализованными и разнообразными. Однако, в большинстве NoSQL СУБД все еще существуют концепции управления доступом.

Аутентификация и авторизация — пользователи могут аутентифицироваться с помощью учетных данных (например, имя пользователя и пароль) или с использованием механизмов аутентификации на основе ключей. После успешной аутентификации пользователь авторизуется для доступа к данным в соответствии с их правами доступа.

Роли — в NoSQL базах данных также существуют концепции ролей, которые могут группировать пользователей и предоставлять им схожие права доступа. Роли могут использоваться для упрощения управления правами доступа и обеспечения согласованности безопасности в системе.

Права доступа — пользователи или роли могут быть наделены различными правами доступа к данным.
Эти права могут включать операции чтения, записи, обновления, удаления и т. д. В зависимости от конкретной NoSQL базы данных, могут также поддерживаться различные уровни гранулярности прав доступа к отдельным объектам данных.

???????? ????? ??????? ? MongoDB

- **read**: позволяет пользователю выполнять операции чтения данных из коллекции.
- **readWrite**: предоставляет пользователю права на выполнение операций чтения и записи данных в коллекцию.
- **dbAdmin**: позволяет пользователю управлять базой данных, включая создание и удаление коллекций, просмотр статистики базы данных и т. д.
- **dbOwner**: предоставляет пользователю полные права доступа ко всем объектам базы данных, включая управление пользователями и ролями.
- **userAdmin**: позволяет пользователю управлять пользователями и ролями в базе данных, включая создание и удаление пользователей и ролей.

- **clusterAdmin**: предоставляет пользователю права на управление кластером MongoDB, включая управление репликацией, шардингом и администрирование кластера.
- **backup**: позволяет пользователю выполнять резервное копирование базы данных MongoDB.
- **restore**: предоставляет пользователю права на восстановление данных из резервной копии базы данных MongoDB.
- **backupAdmin**: позволяет пользователю управлять процессом резервного копирования базы данных MongoDB, включая управление точками сохранения и другими аспектами резервного копирования.

Управление доступом на уровне коллекций/таблиц - во многих NoSQL базах данных можно настраивать права доступа на уровне отдельных коллекций (в MongoDB) или таблиц (в Cassandra). Это позволяет точечно контролировать доступ к конкретным наборам данных в базе данных.

Управление доступом на уровне полей - некоторые NoSQL базы данных также поддерживают гранулярное управление доступом на уровне отдельных полей в документах (например, в MongoDB).

?????? ???????? ???????????? ? MongoDB

```
use mydatabase;

// Создание пользователя с правами чтения и записи для определенной коллекции
db.createUser({
  user: "user1",
  pwd: "password1",
  roles: [
    { role: "readWrite", db: "mydatabase" }
  ]
});

// Создание пользователя с административными правами для базы данных
db.createUser({
  user: "adminUser",
  pwd: "adminPassword",
  roles: [
    { role: "dbAdmin", db: "mydatabase" },
    { role: "userAdmin", db: "mydatabase" }
```

```
  ]
});

// Создание пользователя с правами на выполнение резервного копирования и восстановления
db.createUser({
  user: "backupUser",
  pwd: "backupPassword",
  roles: [
    { role: "backup", db: "mydatabase" },
    { role: "restore", db: "mydatabase" }
  ]
});
```

События создания пользователей в журнале MongoDB mongod.log могут выглядеть следующим образом:

```
2024-03-28T12:34:56.789+0000 I ACCESS   [conn123] Successfully added user: { "user" : "user1", "roles" : [ { "role" : "readWrite", "db" : "mydatabase" } ] }
2024-03-28T12:34:56.790+0000 I ACCESS   [conn123] Successfully added user: { "user" : "adminUser", "roles" : [ { "role" : "dbAdmin", "db" : "mydatabase" }, { "role" : "userAdmin", "db" : "mydatabase" } ] }
2024-03-28T12:34:56.791+0000 I ACCESS   [conn123] Successfully added user: { "user" : "backupUser", "roles" : [ { "role" : "backup", "db" : "mydatabase" }, { "role" : "restore", "db" : "mydatabase" } ] }
```

Поля в логе:

- Дата и время (2024-03-28T12:34:56.789+0000): Метка времени события.
- Уровень журналирования (I ACCESS): Уровень важности сообщения.
- Идентификатор соединения ([conn123]): Идентификатор соединения, в рамках которого была создана запись.
- Сообщение (Successfully added user): Описание события.
- Детали ({ "user" : "user1", "roles" : [...] }): Дополнительная информация о созданном пользователе, включая его имя пользователя (user) и назначенные ему роли (roles).
- Эти записи в журнале сообщают об успешном создании пользователей и предоставлении им соответствующих прав доступа.

?????????????? ???????????

????????? ??????????? ? PostgreSQL

В PostgreSQL шифрование данных можно настроить с использованием различных методов, таких как шифрование на уровне приложения, шифрование на уровне столбцов и шифрование на уровне хранения данных. Ниже представлен обзор этих методов:

1. **Шифрование на уровне приложения**: этот метод предполагает, что данные шифруются на стороне приложения перед тем, как они попадут в базу данных. Приложение самостоятельно обрабатывает шифрование и дешифрование данных перед сохранением их в базе данных или после извлечения из базы данных. Это обеспечивает контроль над процессом шифрования, но требует изменений в коде приложения.
2. **Шифрование на уровне столбцов**: PostgreSQL поддерживает шифрование данных на уровне столбцов с помощью модуля расширения pgcrypto. С использованием этого метода вы можете шифровать конкретные столбцы таблицы, оставляя остальные данные в открытом виде. Преимущество этого метода в том, что шифрование и дешифрование данных происходят автоматически на уровне базы данных, что делает его более удобным для применения.
3. **Шифрование на уровне хранения данных**: этот метод предполагает использование шифрования на уровне хранения данных, когда все данные в базе данных шифруются целиком. PostgreSQL поддерживает шифрование на уровне хранения данных с помощью различных инструментов, таких как Transparent Data Encryption (TDE), который может быть реализован с помощью сторонних расширений или управляемых служб.

Для конфигурации шифрования на уровне столбцов в PostgreSQL с помощью pgcrypto можно использовать следующие шаги:

1. Установите расширение pgcrypto:

```
CREATE EXTENSION pgcrypto;
```

2. Создайте таблицу с зашифрованными столбцами:

```
CREATE TABLE my_table ( id SERIAL PRIMARY KEY, sensitive_data BYTEA ); -- Шифруйте данные
перед вставкой в таблицу
INSERT INTO my_table (sensitive_data) VALUES (pgp_sym_encrypt('my sensitive data',
'secret_key'));
```

3. Дешифруйте данные при их извлечении из таблицы:

```
-- Извлеките зашифрованные данные и дешифруйте их
SELECT pgp_sym_decrypt(sensitive_data, 'secret_key') AS sensitive_data FROM my_table;
```

Для работы с зашифрованными данными в базе данных пользователю обычно требуются следующие права доступа:

Доступ к зашифрованным данным: пользователь должен иметь права на выполнение операций чтения, записи, обновления и удаления данных в зашифрованных таблицах или столбцах. Это обеспечивает пользователю возможность взаимодействовать с данными в базе данных, независимо от их шифрования

```
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE encrypted_table TO user_name;
```

Доступ к ключам шифрования: в некоторых случаях пользователю может потребоваться доступ к ключам шифрования, если он выполняет операции, которые требуют расшифровки данных. Однако доступ к ключам шифрования обычно ограничивается только администраторам базы данных или другим уполномоченным лицам, чтобы предотвратить несанкционированный доступ к ключам.

```
GRANT EXECUTE ON FUNCTION encrypt_function(text) TO user_name;
GRANT EXECUTE ON FUNCTION decrypt_function(text) TO user_name;
```

Доступ к функциям и процедурам: если в базе данных используются функции или процедуры для работы с зашифрованными данными (например, для шифрования или расшифровки), пользователю может потребоваться доступ к этим функциям или процедурам.

```
GRANT EXECUTE ON FUNCTION encryption_function(parameters) TO user_name;
```

Доступ к управлению ключами шифрования (необязательно): в некоторых случаях пользователь может быть уполномочен на управление ключами шифрования или их генерацию. Это может потребоваться, например, при создании новых зашифрованных столбцов или при установке новых ключей шифрования.

```
GRANT CREATE ON SCHEMA encryption_schema TO user_name;
```

????????? ??????????? ? MongoDB

Для настройки шифрования данных в MongoDB вам необходимо выполнить ряд шагов, включая создание ключей шифрования, настройку ключевого провайдера, включение шифрования данных и настройку транспортного шифрования.

1. **Создание ключей шифрования**:
   Пример создания ключей шифрования с использованием `mongocryptd`:
   ```
   mongocryptd --fork --dbpath /var/lib/mongocryptd
   ```
   
2. **Настройка ключевого провайдера**:
   ```
   mongosh use admin db.createCollection("keys") db.keys.insertOne({ "key":
   "<master_key>" })
   ```
   
3. **Включение шифрования данных**:
   Включение шифрования данных при запуске сервера MongoDB:
   ```
   mongod --enableEncryption --encryptionKeyFile /path/to/encryption.key
   ```
   
4. **Включение транспортного шифрования**:
   Пример настройки SSL/TLS для транспортного шифрования:
   ```
   net: ssl: mode: "requireSSL" PEMKeyFile: "/path/to/server.pem" CAFile:
   "/path/to/ca.pem"
   ```

????????? ???????????

????????? ??????????? ? PostgreSQL

Для настройки резервного копирования в PostgreSQL вы можете использовать инструменты резервного копирования, такие как `pg_dump` и `pg_dumpall`, а также встроенную функциональность PostgreSQL для создания резервных копий и восстановления данных.

Ниже шаги, которые обычно выполняются для настройки резервного копирования:

1. **Использование pg_dump для создания резервной копии базы данных:**
   - Используйте команду `pg_dump` для создания текстовых файлов SQL, содержащих структуру базы данных и данные.
   - Пример команды для создания резервной копии базы данных:

     ```
     pg_dump -U username dbname > backup.sql
     ```

     

2. **Использование pg_dumpall для создания резервной копии всех баз данных:**
   - Команда `pg_dumpall` позволяет создавать резервные копии всех баз данных, доступных для указанного пользователя.
   - Пример команды для создания резервной копии всех баз данных:

     ```
     pg_dumpall -U username > backup.sql
     ```

     

?????????? ???????????? ? MongoDB

Для настройки резервного копирования в MongoDB можно использовать инструменты резервного копирования, такие как `mongodump`, а также сторонние инструменты управления резервными копиями, например, MongoDB Atlas Backup или другие инструменты для автоматизации и управления резервным копированием. Ниже настройка резервного копирования в MongoDB с использованием `mongodump`:

1. **Использование mongodump для создания резервных копий:**
   - Пример команды для создания резервной копии всех баз данных MongoDB:

     ```
     mongodump --host <hostname> --port <port> --username <username> --password
     <password> --out <backup_directory>
     ```

     

     Эта команда использует подключение к MongoDB с указанными параметрами (хост, порт, имя пользователя, пароль) и сохраняет резервную копию в указанном каталоге.

Общие рекомендации:

1. **Автоматизация резервного копирования:**
   - Для регулярного создания резервных копий вы можете использовать утилиты планировщика задач, такие как cron в UNIX-подобных системах или Task

Scheduler в Windows.
   - Создайте скрипт, который будет выполнять команду резервного копирования с необходимыми параметрами, а затем настройте его выполнение по расписанию с помощью утилиты планировщика задач.
2. **Хранение резервных копий:**
   - Убедитесь, что резервные копии хранятся в безопасном месте, доступном только авторизованным пользователям.
   - Рассмотрите возможность хранения резервных копий в облачном хранилище или на отдельном сервере с репликацией данных для обеспечения долгосрочного хранения и защиты от потери данных.
3. **Тестирование восстановления:**
   - Регулярно проверяйте процесс восстановления данных из резервной копии, чтобы убедиться, что он работает правильно, и данные могут быть восстановлены в случае чрезвычайной ситуации.

???????? ?????

?? ??????? PostgreSQL

Рассмотрим сценарий, при котором злоумышленник уже имеет доступ к учетной записи с правом `CREATE ROLE` . Пользователь с правом `CREATE ROLE` в PostgreSQL может создать другого пользователя и назначить ему права на редактирование/удаление/просмотр всех таблиц в базе данных.

1. **Создание нового пользователя:** Пользователь с правом `CREATE ROLE` может создать нового пользователя с помощью команды `CREATE ROLE` и назначить ему необходимые права доступа.

   ```
   CREATE ROLE coolhecker LOGIN PASSWORD 'password';
   ```

   

2. **Назначение прав доступа:** чтобы дать новой учетной записи права на редактирование таблиц в БД, ему можно дать роль `SUPERUSER`

   ```
   ALTER ROLE coolhecker WITH SUPERUSER;
   ```

   

   Или можно назначить отдельные привилегии для пользователя, указав необходимые права доступа, например:

   ```
   GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO coolhecker;
   ```

Этот запрос предоставит пользователю `coolhecker` права на просмотр, вставку, обновление и удаление данных из всех таблиц в схеме `public`.

Как это будет видно в логе:

```
2024-03-29 12:00:00.123 UTC [23345] LOG:  statement: CREATE ROLE coolhecker LOGIN PASSWORD 'ehehehehe';
2024-03-29 12:00:01.234 UTC [23345] LOG:  statement: ALTER ROLE coolhecker WITH SUPERUSER;
```

В первой записи выполняется команда `CREATE ROLE`, создающая нового пользователя с именем coolhecker и паролем ehehehehe. Во второй записи выполняется команда `ALTER ROLE`, назначающая пользователю coolhecker права SUPERUSER.

Важно! Для того чтобы увидеть эти записи в логе, необходимо настроить журналирование в соответствии с требуемым уровнем детализации и включить запись административных действий в лог.

Уже на этом этапе злоумышленника можно остановить, если вовремя среагировать на событие создания пользователя с ролью superuser. Как может выглядеть правило sigma для обнаружения такого события:

```
title: Обнаружение создания суперпользователя в PostgreSQL
id: postgresql_create_superuser
status: experimental
description: Обнаружение операции создания пользователя с ролью суперпользователя в
PostgreSQL.
author: [Your Name]
date: 2024-03-31
logsource:
  product: postgresql
detection:
  selection:
    crosstheme1.username: ["!~^(?:postgres)$"]  # Исключаем создание роли "postgres"
    crosstheme1.statement: ["CREATE ROLE%", "%SUPERUSER%"]
  condition: selection
fields:
  - username
  - statement
falsepositives:
  - Легитимные операции создания пользователей с правами суперпользователя, произведенные
```

```
администраторами.
level: high
```

Предположим, что злоумышленника никто не остановил и он идет дальше. Какие операции он может выполнить:

Зайти в БД под созданной учетной записью

В логе такое событие будет выглядеть так:

```
2024-03-31 10:15:00.123 UTC [23345] LOG:  connection authorized: user=coolhecker database=my_database
```

Вывод перечня таблиц в базе данных

```
SELECT table_name FROM information_schema.tables WHERE table_schema = 'public';
```

Этот запрос выбирает все имена таблиц из схемы `public` в базе данных.

Как это будет выглядеть в логе:

```
2024-03-31 10:15:00.123 UTC [23345] LOG:  statement: SELECT table_name FROM information_schema.tables WHERE table_schema = 'public';
```

Вывод названий столбцов для одной из таблиц

Допустим, у нас есть таблица `users`, и злоумышленник хочет увидеть названия ее столбцов.

```
SELECT column_name FROM information_schema.columns WHERE table_schema = 'public' AND table_name = 'users';
```

Этот запрос выбирает все имена столбцов для таблицы `users` из схемы `public` в базе данных.

Как это будет выглядеть в логе:

```
2024-03-31 10:16:00.234 UTC [23346] LOG:  statement: SELECT column_name FROM information_schema.columns WHERE table_schema = 'public' AND table_name = 'users';
```

Выполнить очень широкий запрос

```
SELECT * FROM users;
```

Как это будет выглядеть в логе:

```
2024-03-31 10:16:00.234 UTC [23346] LOG:  statement: SELECT * FROM my_table;
```

Пример правила Sigma, которое обнаруживает 2 последовательных события: авторизацию пользователя в базе и выполнение широкого запроса:

```
title: Обнаружение последовательности аутентификации и выполнения запроса SELECT * FROM в PostgreSQL
id: postgresql_login_followed_by_select
status: experimental
description: Обнаружение события, когда пользователь авторизуется в базе данных, а затем выполняет запрос SELECT * FROM.
author: [Your Name]
date: 2024-04-01
references:
  - https://example.com/postgresql-logging-docs
logsource:
  product: postgresql
detection:
  selection1:
    crosstheme1.event: "connection authorized"
  selection2:
    crosstheme1.statement: "SELECT * FROM"
  timeframe: 1m # Временное окно для обнаружения двух последовательных событий (1 минута)
  condition: selection1 and selection2
fields:
  - user
  - event
  - statement
falsepositives:
  - Легитимные операции, произведенные аутентифицированными пользователями.
level: high
```

?????? ? MongoDB

Рассмотрим аналогичный пример для MongoDB.

Злоумышленник создает учетную запись с широкими правами:

```
use admin

db.createUser({
    user: "adminUser",
    pwd: "adminPassword",
    roles: [
        { role: "root", db: "admin" },
        { role: "readWriteAnyDatabase", db: "admin" },
        { role: "dbAdminAnyDatabase", db: "admin" },
        { role: "userAdminAnyDatabase", db: "admin" }
    ]
})
```

- root: Дает полный административный доступ ко всей системе MongoDB.
- readWriteAnyDatabase: Позволяет читать и записывать данные в любую базу данных.
- dbAdminAnyDatabase: Позволяет управлять базами данных на уровне администратора.
- userAdminAnyDatabase: Позволяет управлять пользователями и их правами на уровне администратора.

Пример правила Sigma, которое будет обнаруживать такое событие:

```
title: Обнаружение создания пользователя с широкими правами в MongoDB
id: mongodb_create_user_with_wide_privileges
status: experimental
description: Обнаружение события, когда создается пользователь с широкими правами в MongoDB.
author: [Your Name]
date: 2024-04-02
```

```
logsource:
  product: mongodb
detection:
  selection:
    crosstheme1.operation: CREATE_USER
    crosstheme1.roles: ["root", "readWriteAnyDatabase", "dbAdminAnyDatabase",
"userAdminAnyDatabase"]
  condition: selection
fields:
  - user
  - roles
falsepositives:
  - Легитимные операции создания пользователей с широкими правами, произведенные
администраторами.
level: high
```

Допустим, злоумышленник решил выбрать все возможные данные из БД. В логе это будет выглядеть так:

```
2024-04-02T12:34:56.789+0000 I COMMAND  [conn1234] command my_database.my_collection command: find { find:
"my_collection", filter: {}, $db: "my_database" } planSummary: COLLSCAN keysExamined:0 docsExamined:10000
cursorExhausted:1 numYields:78 nreturned:10000 reslen:2687313 locks:{ Global: { acquireCount: { r: 158 } },
Database: { acquireCount: { r: 79 } }, Collection: { acquireCount: { r: 79 } } } protocol:op_msg 33ms
```

- `2024-04-02T12:34:56.789+0000` - дата и время события.
- `I COMMAND` - тип команды (в данном случае, команда).
- `[conn1234]` - идентификатор соединения.
- `command my_database.my_collection` - команда, которая была выполнена (в данном случае, поиск документов в коллекции `my_collection` базы данных `my_database`).
- `keysExamined:0` - количество ключей, проанализированных при выполнении запроса (в данном случае, 0).
- `docsExamined:10000` - количество документов, просмотренных при выполнении запроса (в данном случае, 10000).
- `cursorExhausted:1` - флаг, указывающий, что курсор исчерпан.
- `numYields:78` - количество раз, когда сервер передал управление другим операциям (yielded).
- `nreturned:10000` - количество документов, возвращенных в результате запроса (в данном случае, 10000).
- `reslen:2687313` - размер ответа в байтах.
- `locks` - информация о блокировках, используемых запросом.

- `protocol:op_msg` - протокол, используемый для передачи сообщений (в данном случае, op_msg).
- `33ms` - время выполнения запроса (в данном случае, 33 миллисекунды).

Пример правила Sigma, которое будет обнаруживать такое событие:

```
title: Обнаружение выбора всех данных из базы данных MongoDB
id: mongodb_select_all_data
status: experimental
description: Обнаружение события, когда пользователь выбирает все данные из базы данных
MongoDB.
author: [Your Name]
date: 2024-04-03
logsource:
  product: mongodb
detection:
  selection:
    crosstheme1.command: "find"
    crosstheme1.docsExamined: { ">": 0 }
    crosstheme1.nreturned: { ">": 0 }
    crosstheme1.keysExamined: 0
    crosstheme1.cursorExhausted: 1
  condition: selection
fields:
  - user
  - database
  - collection
  - command
falsepositives:
  - Легитимные запросы, которые выбирают все данные из базы данных MongoDB.
level: high
```

- `title`: Название правила.
- `id`: Уникальный идентификатор правила.
- `status`: Статус правила (например, "экспериментальный").
- `description`: Описание того, что делает правило.
- `author`: Ваше имя или имя автора правила.
- `date`: Дата создания правила.
- `logsource`: Источник журналов, в данном случае - продукт `mongodb`.

- `detection`: Определение, которое указывает, как обнаружить событие. Мы используем выборку по нескольким полям, таким как `command` (команда), `docsExamined` (количество документов, просмотренных при выполнении запроса), `nreturned` (количество возвращенных документов), `keysExamined` (количество ключей, проанализированных при выполнении запроса) и `cursorExhausted` (флаг, указывающий, что курсор исчерпан).
- `condition`: Условие, которое определяет, что событие соответствует правилу.
- `fields`: Поля, которые будут отображены в результате обнаружения.
- `falsepositives`: Возможные ситуации, которые могут быть ложноположительными.
- `level`: Уровень серьезности обнаружения (например, "высокий").

Revision #1
Created 17 October 2025 12:50:54 by Admin
Updated 17 October 2025 12:54:28 by Admin