PG BootCamp Russia 2024 Kazan

PGBootCamp.ru

Анонимизация данных с использованием pg_anon

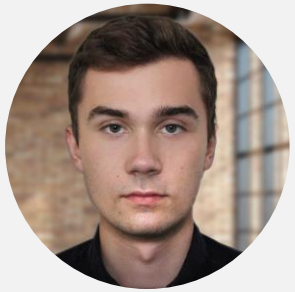
Максим Ибрагимов, старший разработчик «Тантор Лабс»
Денис Родионов, разработчик «Тантор Лабс»

tantor

Немного о нас

Максим Ибрагимов

Окончил ПХТТ по специальности «Программирование в компьютерных системах». Интересуюсь технологиями с раннего возраста. Уже более 10 лет занимаюсь проектированием, сопровождением и оптимизацией баз данных.



Денис Родионов

Учусь в МГТУ имени Баумана по специальности "Компьютерная безопасность" на 4 курсе. В начале 2023 года присоединился к команде "Тантор Лабс" разработчиком СУБД. В течение последнего года занимаюсь разработкой утилит для PostgreSQL.

О чем мы расскажем?

- Что такое маскирование и какую задачу оно решает
- Как реализовать маскирование, используя только PostgreSQL
- Как мы его автоматизировали
- Продемонстрируем работу утилиты
- Ответим на вопросы

Маскирование. Что это и зачем?

Маскирование данных — это процесс замены конфиденциальной информации на случайно сгенерированные данные, которые не позволяют идентифицировать исходные значения.

Свойства:
- соблюдение целостности данных
- сохранение формата данных
- надежность

Цель: иметь обезличенную базу данных, с которой можно работать так же, как и с исходной базой.

Области безопасного применения

Разработка

Воспроизведение сложных проблем, возникших на производственном окружении

Тестирование

Проведение различных тестов, в т.ч. нагрузочных, с реальным объемом данных

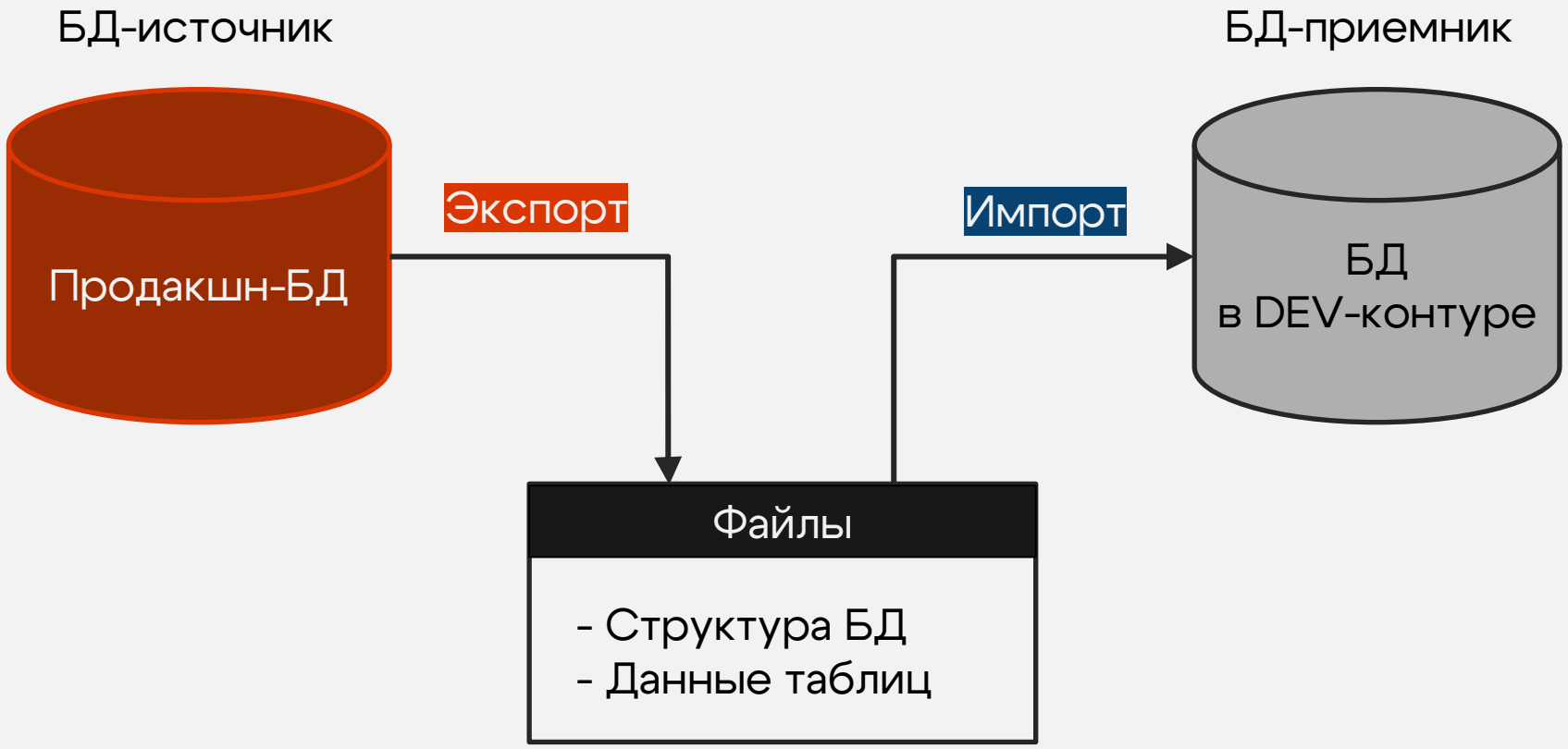
ИБ

Определение потенциальных уязвимостей

Аналитика, маркетинг

Исследование обезличенных данных, для выведения статистики

Общий процесс



Экспорт = dump («дамп» то, что делает утилита PostgreSQL pg_dump)
Импорт = restore («рестор» то, что делает утилита PostgreSQL pg_restore)

Иллюстрация процесса командами SQL – Экспорт [1]

Создание таблицы, содержащей конфиденциальные данные:

```
-- Создание таблицы-источника
create table users (
    id bigserial,
    email text,
    login text
);
```

```
pg_bootcamp=# select * from users;

 id |  email  | login
----+---------+-------

(0 строк)
```

Иллюстрация процесса командами SQL – Экспорт [2]

Наполнение таблицы конфиденциальными данными:

```
-- Наполнение таблицы-источника
insert into users (email, login)
select
 'user' || generate_series(1001, 1020) || '@example.com',
 'user' || generate_series(1001, 1020);
```

```
pg_bootcamp=# select * from users;

 id |          email       |  login
----+----------------------+----------
  1 | user1001@example.com | user1001
  2 | user1002@example.com | user1002
  3 | user1003@example.com | user1003
...
```

Иллюстрация процесса командами SQL – Экспорт [3]

Экспорт данных «как есть»:

```
-- из таблицы-источника в CSV файл (без маскирования)
copy (
        select *
        from users
) to '/tmp/users.csv' with csv;
```

```
$ cat /tmp/users.csv

1,user1001@example.com,user1001
2,user1002@example.com,user1002
3,user1003@example.com,user1003
4,user1004@example.com,user1004
...
```

Иллюстрация процесса командами SQL – Экспорт [4]

Просмотр данных с применением фильтра (правила маскирования):

```
pg_bootcamp=# select
    id,
    md5(email) || '@abc.com' as email, -- хэширование email
    login
from users;

 id |                email                |  login
----+-------------------------------------+----------
  1 | 385513d80895c4c5e19c91d1df9eacae@abc.com | user1001
  2 | 9f4c0c30f85b0353c4d5fe3c9cc633e3@abc.com | user1002
  3 | e4e9fe7090f5be634be77db8f86e453c@abc.com | user1003
...
```

Иллюстрация процесса командами SQL – Экспорт [5]

Экспорт данных с применением фильтра (правила маскирования):

```
-- из таблицы-источника в CSV файл (с маскированием)
copy (
  select
    id,
    md5(email) || '@abc.com' as email, -- хэширование email
    login from users
) to '/tmp/users_anonymized.csv' with csv;
```

```
$ cat /tmp/users_anonymized.csv

1,385513d80895c4c5e19c91d1df9eacae@abc.com,user1001
2,9f4c0c30f85b0353c4d5fe3c9cc633e3@abc.com,user1002
3,e4e9fe7090f5be634be77db8f86e453c@abc.com,user1003
...
```

Иллюстрация процесса командами SQL – Импорт [1]

Создание таблицы-приемника:

```
create table users_anonymized (
    id bigserial,
    email text,
    login text
);
```

```
pg_bootcamp=# select * from users_anonymized;

 id | email | login
----+-------+-------

(0 строк)
```

Иллюстрация процесса командами SQL – Импорт [2]

Импорт данных в таблицу-приемник:

```
copy users_anonymized
from '/tmp/users_anonymized.csv'
with csv;
```

```
pg_bootcamp=# select * from users_anonymized;

 id |                email                |  login
----+-------------------------------------+----------
  1 | 385513d80895c4c5e19c91d1df9eacae@abc.com | user1001
  2 | 9f4c0c30f85b0353c4d5fe3c9cc633e3@abc.com | user1002
  3 | e4e9fe7090f5be634be77db8f86e453c@abc.com | user1003
...
```

Автоматизация маскирования

Как сделать дамп с маски-рованием

1

Дамп структуры (pre-data)

- Используя pg_dump, выгрузить структуру БД

2

Дамп ограничений и индексов (post-data)

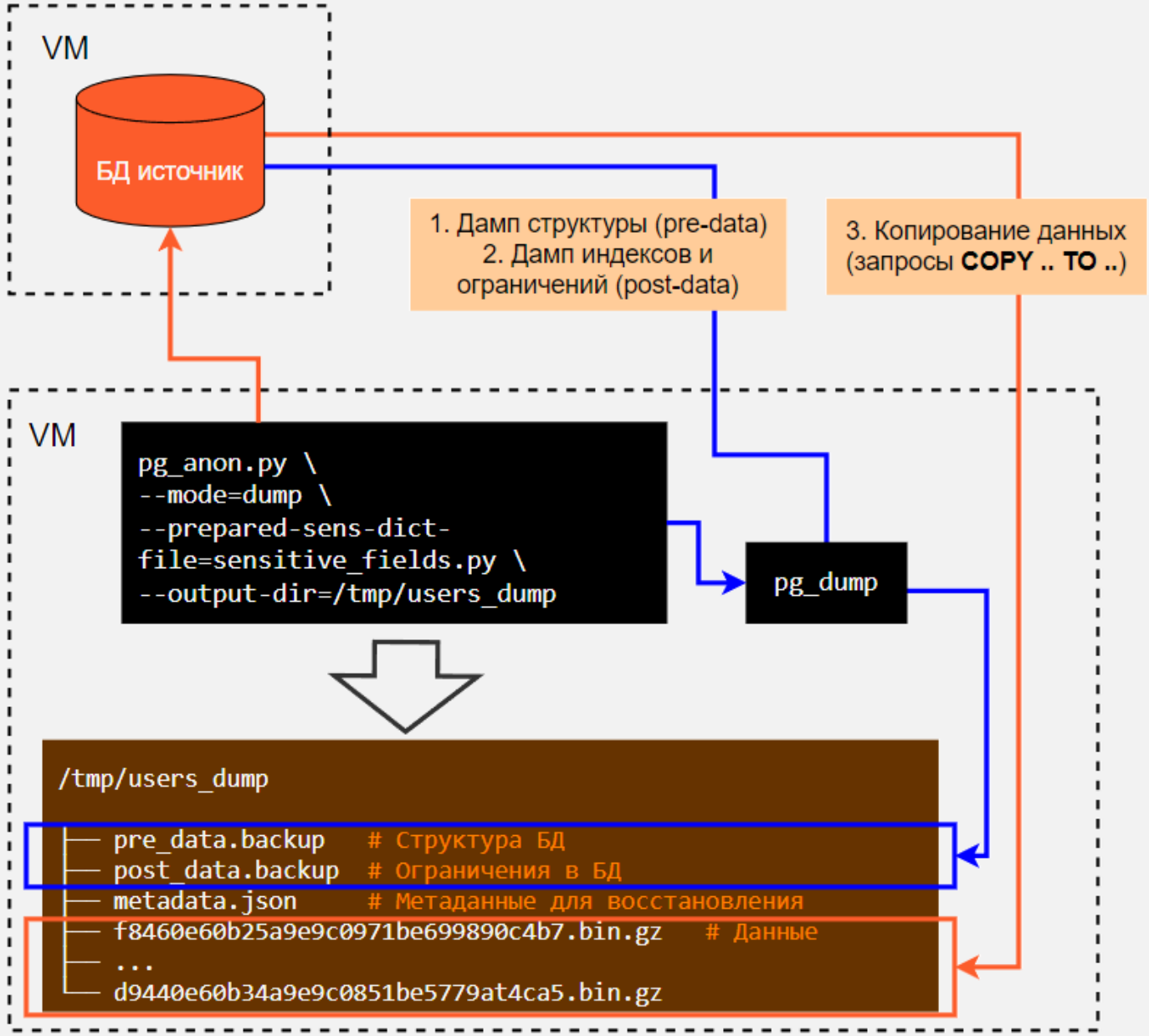
- Используя pg_dump, выгрузить ограничения индексы в БД

3

Дамп данных

- Подготовить запросы SELECT … FROM …, в которых указать функции для маскирования
- Используя подготовленные SELECT вместе с COPY … TO …, выгрузить данные в файлы

Дамп данных БД источника



Дамп данных БД источника: словарь

```
cat sens_fields.py
>>
{
   "dictionary": [
      {
         "schema": "public",
         "table": "users",
         "fields": {
            "email": "md5(email) || '@abc.com'"
         }
      },
   ]
}
```

```
copy (
  select
    id,
    md5(email) || '@abc.com' as email,
    login from users
) to '...' with binary;
```

Как сделать восстановление маскированного дампа

1

Воспроизведение структуры

- Используя pg_restore, воспроизвести структуру БД

2

Загрузка данных

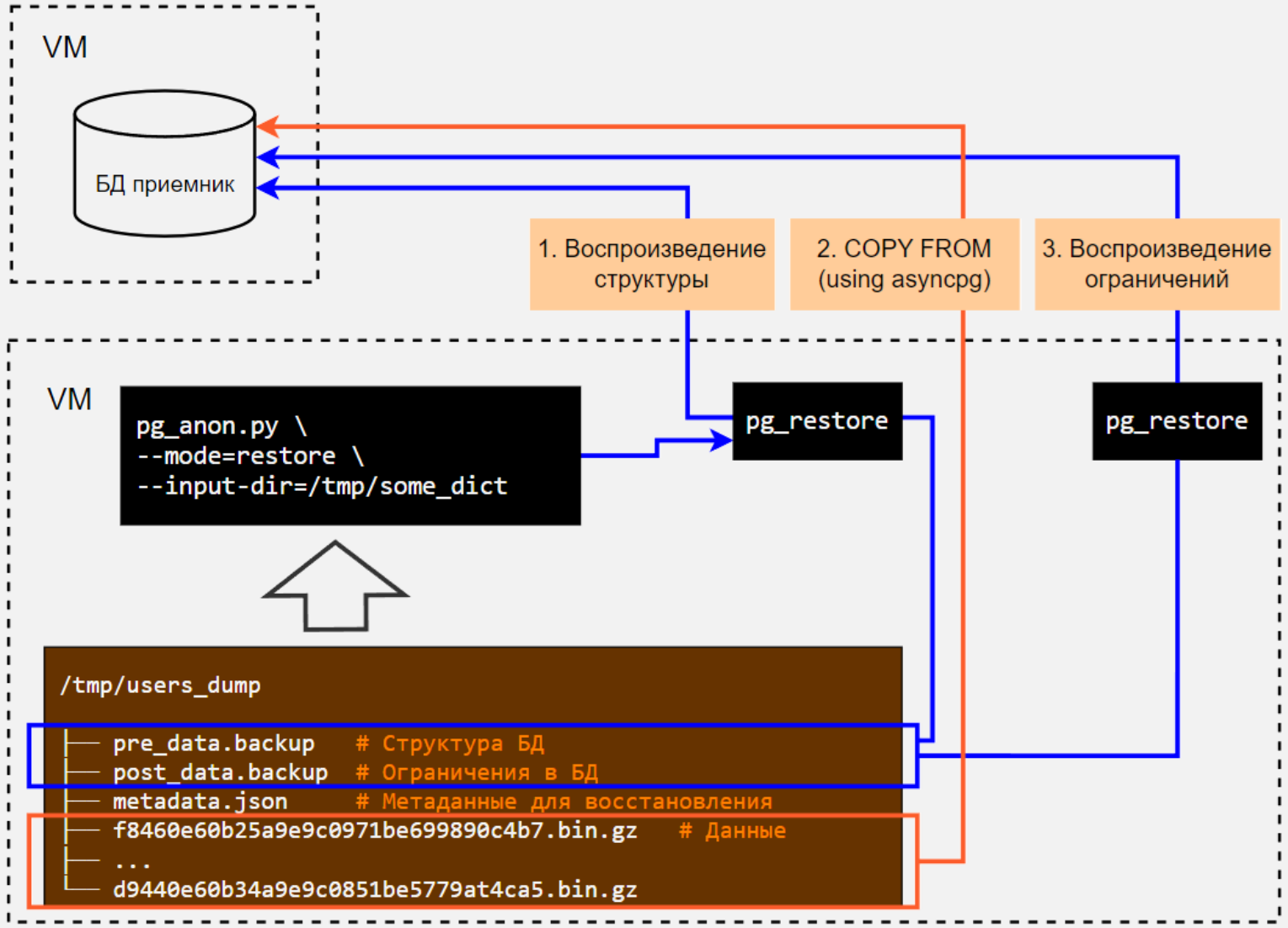
- Используя пачку подготовленных запросов COPY … FROM …, загрузить данные из файлов в таблицы

3

Воспроизведение ограничений и индексов

- Используя pg_restore, воспроизвести ограничения в БД

Рестор данных БД в приемник



Как выполнить сканирование и автоматически создать словарь для дампа

1

Проверка названий полей и их данных

- Сверка названий полей, в соответствии с набором правил, для определения их конфиденциальности
- Сверка данных в полях, в соответствии с набором правил, для определения их конфиденциальности

2

Проверка данных

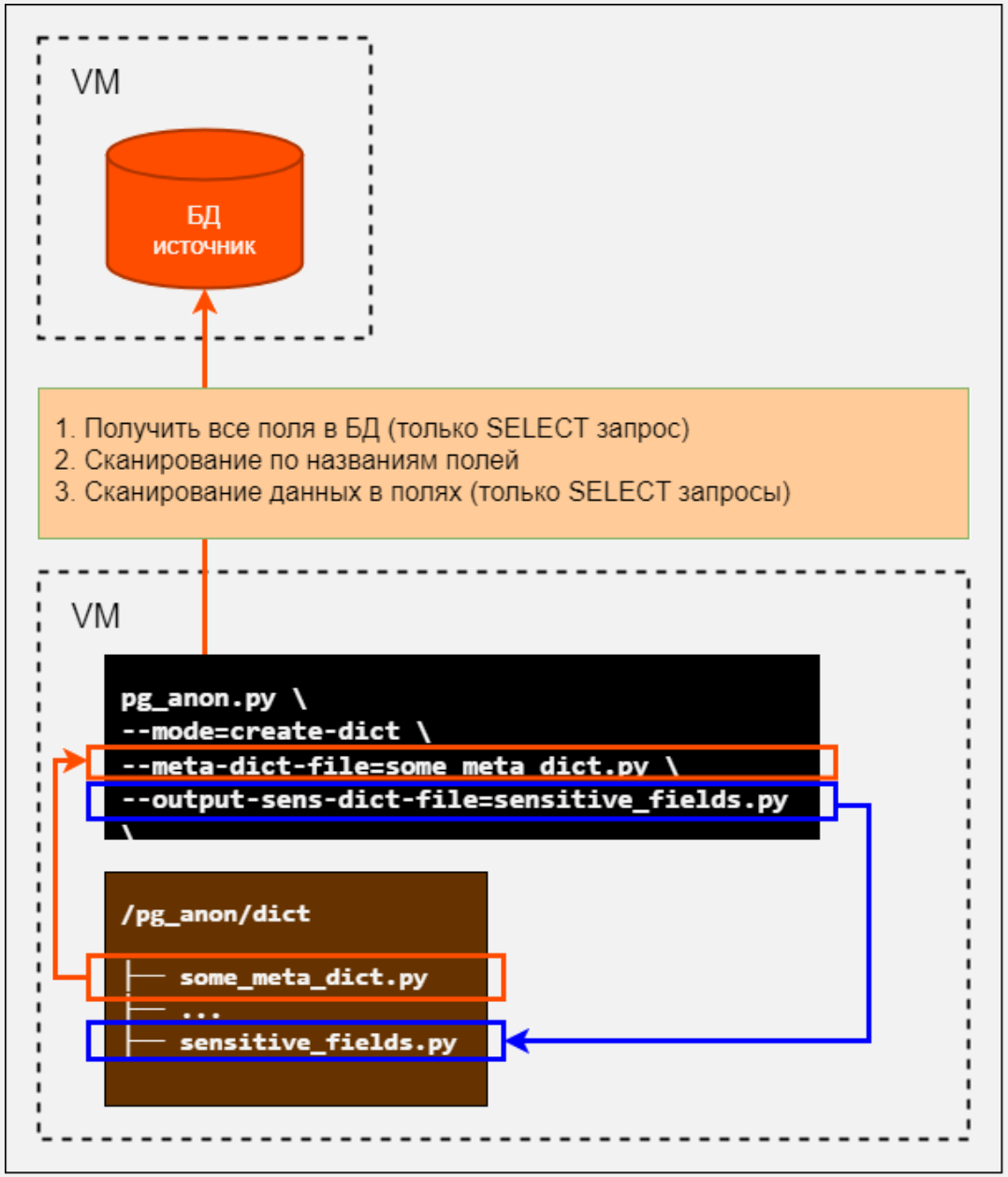
- Определение правил маскирования для найденных конфиденциальных полей

3

Создание словаря для дампа

- Создание словаря с набором правил, для последующего использования в экспорте

Автоматическое создание словаря



Автоматическое создание словаря: мета-словарь

```
cat some_meta_dict.py
>>
{
  "field": {  # Названия полей
    "rules": [
      "^inn"
    ],
  },
  "data_regex": {  # Содержимое полей
    "rules": [
      """.*\@.*""",
    ]
  },
  "funcs": {  # Правила маскирования
    "text": "md5(\"%s\") || '@abc.com'"
    "custom_type": “***"
  }
}
```

```
cat sensitive_fields.py
>>
{
  "dictionary": [
    {
      "schema": "public", # Название схемы
      "table": "users", # Название таблицы
      "fields": { # Маскируемые поля
        "email": "md5(\“email\") || '@abc.com'",
      }
    },
  ]
}
```

Практика

Немного об инициализации

Запуск в режиме **init** создает в указанной БД схему **anon_funcs**

Данная схема содержит функции маскирования:

- **noise** – случайное число
- **dnoise** – случайное значение для дат
- **digest** – хэширование
- **partial** – замена части строки
- **partial_email** – замена части строки email
- **random_string** – случайная строка
- и т.п.

Безопасность и параллельность

- Отсутствуют деструктивные запросы
- Все транзакции дампа (экспорта) работают на уровне **repeatable_read** с флагом **read_only**
- Перед импортом, производится проверка на пустоту базы

- Возможность задать количество процессов
- Возможность задать количество соединений на процесс

О чем мы не рассказали

- Разные режимы дампа в pg_anon
- Разные режимы рестора в pg_anon
- Гибкие правила маскирования
- Метасловарь как инструмент для автоматической генерации словарей
- Словарь несенситивных полей, используемый для ускорения повторного сканирования
- Использование набора словарей при маскировании
- Использование набора метасловарей при сканировании
- Режимы отладки маскирования

Дальнейшее развитие

- Надстройка для декларативного описания процессов переноса данных с поддержкой хуков
- API
- Графический интерфейс
- Поддержка других форматов файлов для словарей

Спасибо!

Максим Ибрагимов, Денис Родионов, «Тантор Лабс»

tantor