УДК 004.056
DOI: 10.31673/2409-7292.2022.032431

Баргилевич О.А., Найман Г.Г.

ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ PostgreSQL

В даній статті проведено аналіз системи управління базами даних PostgreSQL сучасного підприємства. Також розглянуто базовий концепт бази даних, їх види, підвиди та прогресивне розширення бази даних в СУБД. Проаналізовано основні налаштування задля поліпшення безпеки СУБД, що мали можливі шляхи покращення безпеки як фізичного, так і віртуального середовища. Також проведений практичний приклад отримання доступу до середовища та безпосередньо до даних СУБД при базовій конфігурації, зроблені відповідні висновки та нотатки про можливі шляхи їх вирішення. Розроблено рекомендації, що бажано застосувати для покращення стану безпеки СУБД сучасного підприємства та уникнення випадків втрати або поширення конфіденційних даних.

Ключові слова: база даних, СУБД, PostgreSQL, безпека даних.

Вступ

Дані зберігаються в великих базах даних та містять десятки колекцій, видів та підвидів. Зберігання даних в мережі стала невід'ємною частиною її функціональності заради зручності та простоти їх використання. Однак, більшість даних являються досить персональними. Наприклад, дані кредитних фізичних карток, номери телефонів, електронних адрес, конфіденційні дані по документам. Вже відомі випадки та обставини при яких такі дані потрапляли у відкритий доступ, чим спричиняли великий дискомфорт та можливі проблеми, адже зловмисники можуть на свій розсуд користуватися конфіденційними даними. Саме тому, база даних повинна являтися самим захищеним місцем для зберігання, щоб інформація в ній не потрапила в чужі руки.

Бази даних здебільше розміщуються на окремих серверах, звідкіля різні сервіси можуть маніпулювати даними в ній, через те потрібно грамотно все налаштувати не тільки в рамках бази даних, але й на фізичному комп'ютері, де вона розміщуються. Вже відомі спеціальні додатки, які дозволяють дізнаватися як заволодіти авторизаційними даними для зв'язку з сервером третім особам. Достатньо розповсюджене явище коли в межах інтернету можливо зустріти сервіси, що використовують базові налаштування бази даних та серверного доступу, що є в корні не правильним явищем. Самий простий приклад, це адміністратори не змінюють базовий пароль «admin» для підключення.

Дана робота спрямована з'ясувати та надати рекомендації по потрібному налаштуванню баз даних, фізичних комп'ютерів на яких вони розміщенні. За допомогою наданих рекомендацій та аналізу можливо буде налаштувати середовище для потенційного зменшення несанкціонованого доступу, розповсюдження конфіденційних даних та ознайомитися технологією захисту в цілому.

Мета роботи – розробити порядок застосування захисту СУБД PostgreSQL сучасного підприємства та рекомендації щодо його реалізації.

Забезпечення безпеки СУБД PostgreSQL

Захист даних є надзвичайно важливим для успіху будь-якого підприємства, а також для безпеки його клієнтів. Перша частина розробки рекомендацій буде стосуватися огляду як підключатися на сервер та на скільки він буде доступний з точки зору безпеки. При конфігурації безпеки потрібно відштовхуватись від принципу найменших привілеїв (тобто потрібно якісно обдумати та дозволити таку кількість доступів скільки буде рахуватися потрібно, та ні в якому разі не більше).

Спочатку надамо рекомендації щодо фізичного доступу. Першим кроком фізичний доступ слід максимально обмежити та переконатись, що сервер на якому розташована СУБД знаходиться у захищеному місці. Під захищеним місцем мається на увазі приватна серверна кімната. В разі окремої кімнати для серверу можна вжити заходи для забезпечення того, щоб у приміщення міг увійти лише уповноважений персонал, вхід в дану кімнату був проведений

лише через індивідуальну ключ-карту, а також використовувався моніторинг, наприклад, відеоспостереження. У разі якщо є потреба в використовуванні суспільних базових просторів, то потрібно впевнитися, що постачальник має достатньо потрібну політику безпеки, та не дає поширенню несанкціонованому доступу. Однак з хмарними просторами СУБД має достатньо малу кількість налаштувань. Однак якщо користуватися надійними постачальниками на зразок Amazon або Google то можна бути впевненим що вони надають високий рівень фізичної безпеки. Але для хмарних та об'єктів спільного розміщення важливо впевнитися в наявності документації, що підтверджує рівень безпеки.

Наступним етапом рекомендацій є підключення. Хоча PostgreSQL використовує один сокет для з'єднання, за допомогою налаштування параметра конфігурації «unix_socket_directories», він може створювати декілька сокетів, з яких може мати різні дозволи кожен. Однак потрібно дуже обережно його налаштувати для повноцінного контролю.

Розглянемо налаштування роз'єму TCP/IP. Використання мережевого сокета TCP/IP потрібно, для того, щоб з віддаленої системи отримати доступ до свого сервера PostgreSQL. Зведення до мінімуму зону потенційної атаки для тих, хто намагається отримати доступ до системи, залежить від розміщення серверу в мережі. Коли сервер всередині корпоративної мережі, він може бути розміщений у кількох фізичних мережах або внутрішній системі VLAN. Налаштувати систему потрібно лише на прослуховування та прийняття з'єднань у мережах. Щоб бути впевненим, що PostgreSQL прослуховує та приймає підключення тільки до потрібної мережі, потрібно використовувати конфігураційний параметр «listen_addresses» у «postgresql.conf» [1].

Також налаштування Брандмауера є досить важливим. Брандмауери важливі інструменти для уникнення доступу з неавторизованих джерел до мережевих портів. Вони також пропонують засоби ведення журналу, які використовують для більш активного виявлення спроб вторгнення. На прикладі звичайного брандмауера ви можете сформувати правила на вході та виході, які визначають дозволений трафік. Ці правила формуються на основі параметрів, які необхідно заповнити:

задати протокол (TCP, IPv6) ;

порт на якому розташовується PostgreSQL.

IP-адреса або доменне ім'я, місця з якого відбувається підключення. Для поліпшення налаштувань захисту, деякі брандмауери пропонують додаткові функції. Для мінімізування доступу до PostgreSQL, ми можемо створити правило: Щоб трафік TCP (IPv6), який надходить з адреси не наших сервер додатків до порту, на якому розташовується PostgreSQL, відхилявся (або залишався чорним).

Бажано також створити правила для вихідних даних, якщо на сервері вашому встановлені обгортки іноземних даних або подібні розширення. При використанні хмарних технологій рекомендується використовувати вбудовані в хмарну платформу брандмауери, тому що вони дозволяють створювати правила, які можна неодноразово використовувати та підключати до кількох серверів, цим самим значно полегшують керування.

Визначимо налаштування для шифрування транспорту. Зазвичай трафік до сервера бази даних, який протікає через мережу, потрібно шифрувати. OpenSSL, який використовується в PostgreSQL, забезпечує безпечну переправу даних та бере за основу принцип шифрування TLS (рис. 1).

Щоб у PostgreSQL зашифрувати дані потрібні ключ та сертифікат сервера. Вони вказуються за допомогою «ssl_key_file» і «ssl_cert_file». Рекомендується їх захищати парольною фразою, вона вводиться вручну або за допомогою сценарію, який отримує її від імені сервера, за допомогою конфігурації «ssl_passphrase_command». Якщо в наявності вже є ЦС (центр сертифікації), є можливість використовувати сертифікати з PostgreSQL. Через налаштування конфігурації «ssl_ca_file» і «ssl_crl_file» зможуть надати СУБД сертифікати що в свою чергу надає гнучкість відкликати сертифікати у відповідь на інциденти безпеки та дозволити серверу відхиляти сертифікати клієнта або клієнтські сертифікати сервера. Це

також дозволяє налаштувати клієнта та сервера так, щоб вони відхиляли один одного, якщо ідентичність одного з них не може бути перевірена через ланцюг довіри.

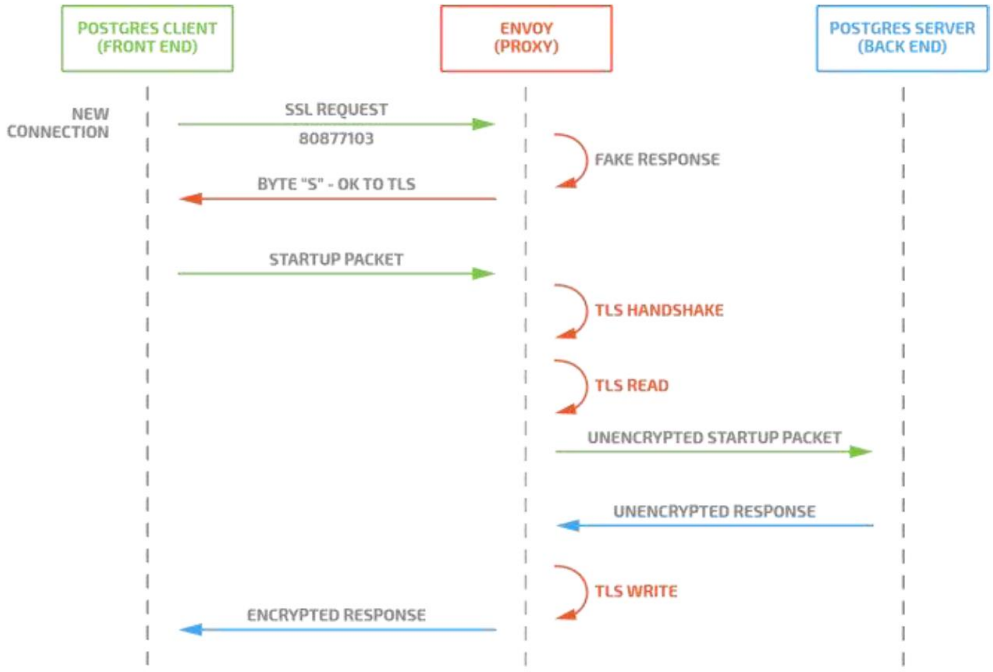

Рис. 1. Процедура виконання шифрування за принципом TLS [2]

В першу чергу потрібно переконатися у безпечному використанні TLS. Рекомендується перевірити та налаштувати такий ряд параметрів конфігурації у файлі конфігурації:

«*postgresql.conf*»;
«*ssl_ciphers*»;
«*ssl_ecdh_curve*»;
«*ssl_dh_params_file*»;
«*ssl_min_protocol_version*».

Автентифікація клієнта це важливий елемент безпеки, тому ми надамо рекомендації як налаштувати конфігурацію таким чином щоб користувачі автентифікувалися та повідомлялося чи успішно вони під ’єдналися до сервера через файл конфігурації «pg_hba.conf».

Файл конфігурації «pg_hba.conf», знаходиться в каталозі даних PostgreSQL. Він визначає методи автентифікації та правила доступу для сервера даних. Послідовність рядків виконується при підключенні (з’єднанні), тоді як перший рядок визначає автентифікації, який буде використовуватися. Всього в наявності є 7 різних форматів рядків у файлі [3]. З них є три основні варіанти. Решта є доповнюючими полями конфігурації. Потрібно налаштувати файл таким чином щоб обов’язково був в наявності:

тип з’єднання;
ім’я бази даних;
ім’я користувача;
мережеву адресу/підмережу клієнта;
метод автентифікації, та додаткові до автентифікації поля.

Рекомендується в для мережевих використовувати hostssl або hostgssenc (опції для налаштування конфігурації), задля забезпечення шифрування з’єднань.

В СУБД є можливість налаштувати метод автентифікації довіри, однак його потрібно використовувати лише в окремих випадках, оскільки за його допомогою можливо підключитися окремому клієнту без подальшої. Даний метод слід використовувати лише для тестування та випадках розробки тільки на локальну обладнанні, однак тільки при

використанні TLS або коли доступ до віддаленого або локального обладнання мають лише повністю довірені користувачі та безпека даних не є проблемою. Використовувати метод довіри потрібно з особливою обережністю. Цей метод являється дуже ризикованим!

Визначимо правила хешування. Спосіб хешування md використовувався на протязі багатьох та був одним з найпопулярніших способів хешування у зв'язці PostgreSQL та досі користується попитом. Однак рекомендується при налаштуванні використовувати scram-sha-256 якщо потрібна необхідність використання пароля при автентифікації. На даний момент md5 та scram-sha-256 використовують метод відповіді на виклик задля більшої безпеки. А хешовані паролі зберігається на сервері. Їх відмінність полягає в тому що scram-sha-256 зберігає хеші в окремій формі, такий метод рахується більш криптографічно безпечним, для уникнення ситуацій коли злодіяч зможе отримати доступ до хешів.

При ситуації якщо є окремий сервер PostgreSQL та потрібно встановлювати з'єднання з ним при використанні пароля рекомендується використовувати scramsha-256 як метод автентифікації. На даний момент спосіб хешування md5 являється застарілим тож потрібно уникати його використання в нових системах.

Розглянемо інтеграції з системами єдиного входу. LDAP (мережевий протокол для надсилання запитів або модифікації даних) та інші системи часто використовують в корпоративних середовищах при інтеграції з системами єдиного входу [4].

При використанні такого підходу сервер PostgreSQL налаштовується на автентифікацію користувачів через протокол LDAP або інші зв'язні системи. При використанні LDAP є можливість використання різноманітних способів для контроля доступу користувачів (для окремих ролей та учасників різних груп). Якщо використовується LDAP рекомендується в файл «pg_hba.conf» внести додаткові параметри для з'єднання, особливо рекомендується налаштувати фільтр пошуку який дозволяє підключатися до бази даних лише користувачам, які відповідають фільтру. Альтернативним варіантом до LDAP являється Kerberos (протокол що дає можливість автентифікації до встановлення каналу). Налаштування даного протоколу являється більш складним, однак він при правильному налаштуванні вважається більш безпечним. Також він дає можливість налаштувати автоматичну автентифікацію для клієнтського програмного забезпечення, якщо воно має таку підтримку.

Метод автентифікації LDAP користується великою популярністю та має позитивні відгуки, але рекомендується віддати перевагу автентифікації за допомогою Kerberos, оскільки даний протокол ніколи не надсилається на сервер PostgreSQL пароль користувача.

Надалі визначимо налаштування сертифікатів TLS. Рекомендується використовувати сертифікати та шифрування TLS для автентифікації та передачі даних. Автентифікація сертифікатів працює, довіряючи сертифікату верхнього рівня, щоб видавати сертифікати лише довіреним клієнтам. Клієнти, які мають сертифікат і ключ, видані вищим органом, який також видав сертифікат і ключ сервера, можна вважати надійними. Для початку потрібно створити ключ та сертифікат центру сертифікації який був розглянутий раніше. Ці дані достатньо конфіденційні тому тримати їх потрібно в цілковитій. Далі потрібно створити ключ та сертифікат для сервера PostgreSQL та підписати його раніше створеними ключом та сертифікатом від. Ці два файли (сертифікат та ключ) потрібно встановити на сервері PostgreSQL разом з тільки копією сертифіката ЦС (рис. 2).

Також хорошою практикою вважається використовування списку відкликання сертифікатів, це зроблено задля вистежування сертифікатів , котрим більше не можна довіряти. Сертифікати являються найкращим способом на сьогоднішній день для автентифікації автоматизованих систем, котрі мають підключатися через мережу до сервера PostgreSQL [5].

Виведемо застосування додаткової конфігурації. Рекомендується налаштувати додаткові параметри для конфігурації:

«authentication_timeout» - параметр, що встановлюється через файл «postgresql.conf». За допомогою визначення цього параметру можливо встановити максимальну кількість часу, з який автентифікація повинна закінчитися, до того моменту коли сервер закриє з'єднання.

Оскільки для у з'єднання є кінцева кількість місць, даний параметр відключає спроби з'єднання які тривають більше зазначеного часу та займають на невизначений термін;

«auth_delay» - даний модуль можливо завантажити через параметр конфігурації «shared_preload_libraries» у «postgresql.conf». Даний параметр слугує для встановлення паузи між спробами автентифікації. За допомогою цього значно ускладняються атаки методу швидкісного підбору. Наступним важливим налаштуванням є створення та модифікація ролей, котрі мають унікальні права в СУБД.

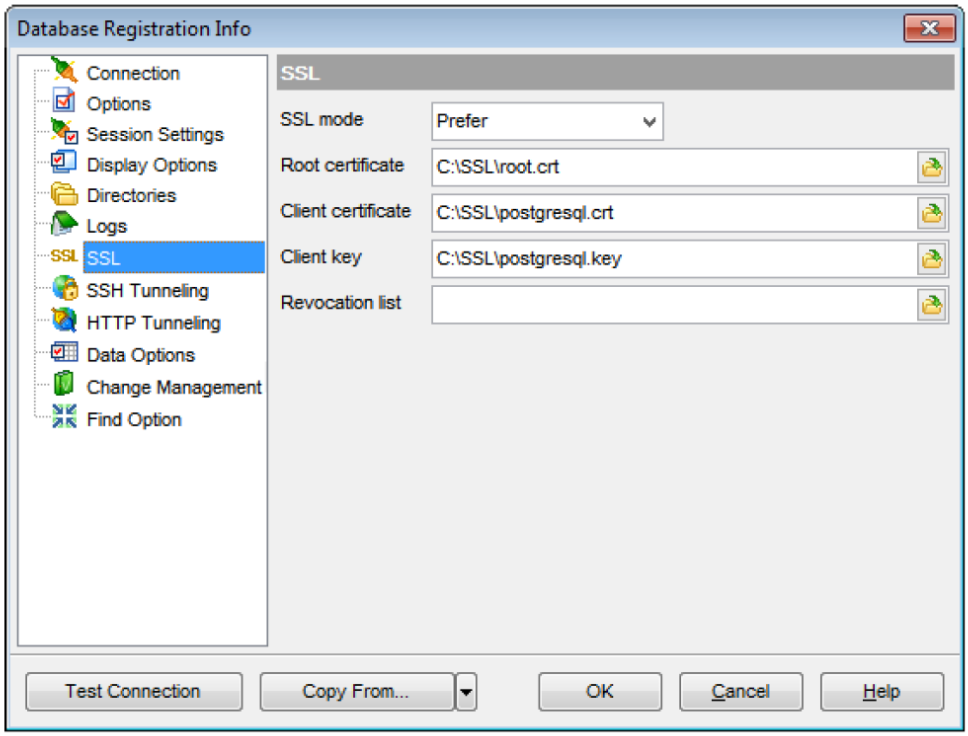

Рис. 2. Додавання сертифікатів та ключа для СУБД

Як ми розглянули раніше роль може бути унаслідувана від інших ролей або членів. У ролей є окрема кількість встановлюємих атрибутів, навіть такі, які роблять їх обліковими записами користувачів. Ролі мають ряд фіксованих атрибутів, які можна встановити:

LOGIN - фактично як зрозуміло з назви є логіном користувача;

SUPERUSER - чи являється цей користувач супер-користувачем який може змінювати фактично все в системі (на зразок власника);

CREATEDB - чи має можливість дана роль створювати бази даних;

CREATEROLE – чи має можливість створювати нові ролі;

REPLICATION – чи може створювати копію бази даних;

PASSWORD – пароль (опціонально);

BYPASSRLS – чи може не проходити перевірки безпеки на рівні рядків;

VALID UNTIL – дійсний до якогось часу (опціонально).

Рекомендується уважно слідкувати кому та при яких обставинах надаються важливі атрибути (SUPERUSER, CREATEDB, CREATEROLE), а також не використовувати для повсякденної роботи привілеї SUPERUSER. Після налаштування ролей буде виконуватись схожа схема як на рис. 3.

Варто мати на увазі що PostgreSQL за базовими налаштуваннями не надає можливостей контроля складності пароля. Але, він має можливість підключити сторонній модуль для такої перевірки. Щоб даний додатковий модуль мав ефект, користувач повинний власноруч не змінювати пароль за допомогою минулого за- хешованого рядка. Гарним вирішенням даної

проблеми являється використовування зовнішнього сервісу для автентифікації, на зразок раніше розглянутих LDAP та Kerberos. Для посилення захисту пароля потрібно використовувати профіль пароля, за допомогою методів автентифікації (md5 або scram-sha-256). Профілі паролів повинні бути зазначені та застосовані до ключових ролей супер-користувачем.

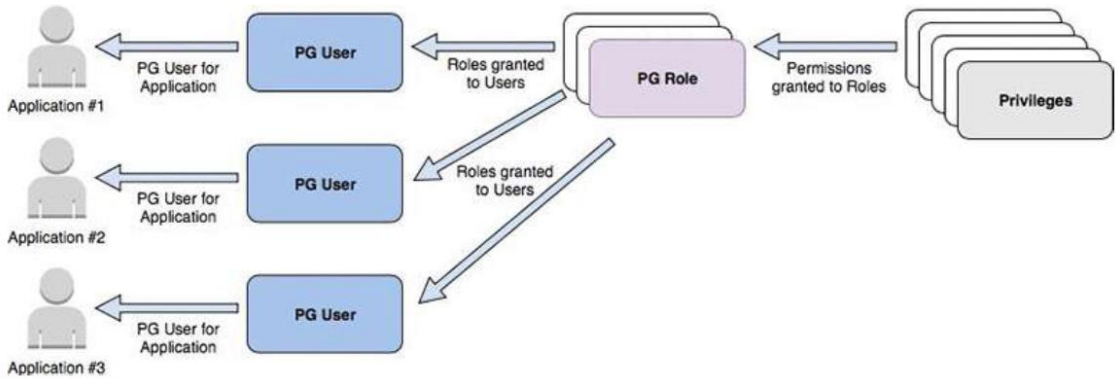

Рис. 3. Схема захищеності за допомогою доступів [6]

Налаштування профілю поєднує такі параметри:

«FAILED_LOGIN_ATTEMPTS» – цифра що визначає кількість невдалих спроб вводу даних авторизації, після чого роль/користувач буде заблокована до часу який зазначений в атрибуті «PASSWORD_LOCK_TIME»;

«PASSWORD_LIFE_TIME» – зазначена кількість часу (в днях), після чого буде рекомендовано змінити пароль;

«PASSWORD_GRACE_TIME» – заданий період за який користувач повинен буде змінити пароль якщо він вийшов за період «PASSWORD_LIFE_TIME». Якщо цей термін скінчився у користувача не буде можливості змінювати будь які дані, буде лише можливість перегляду таблиць;

«PASSWORD_REUSE_TIME» – період, який користувач повинен очікувати перед тим як використовувати минулий пароль;

«PASSWORD_REUSE_MAX» – кількість використовувань різних паролей до того часу як можливо буде використовувати старі;

«PASSWORD_VERIFY_FUNCTION» – назва створеної функції в базі даних котра має можливість перевірити складність введеного паролю;

«PASSWORD_REUSE_MAX» – кількість можливих повторювань паролю при його зміні;

«PASSWORD_ALLOW_HASHED» - чи потрібно забороняти мануально змінювати пароль у вигляді хеша.

Базово PostgreSQL налаштований з поєднанням декількох ролей моніторингу. Вони дають можливість певним ролям надавати додаткові можливості, котрі використовуються для моніторингу системи, що позбавляє необхідності надавати роль супер-користувача:

«pg_monitor» - стандартна роль, яка є батьківською для наступних:

«pg_read_all_settings» - надає можливість читати деякі змінні конфігурації котрі доступні лише супер-користувачам;

«pg_read_all_stats» - дозволяє читати таблиці пов’язані з аудитом та використовувати додаткові розширення що мають зв’язок зі статистикою, навіть такі які можуть бачити та використовувати лише супер-користувачі;

«pg_stat_scan_tables» - приводити до дії функції сканування та моніторингу, навіть такі, які спроможні блокувати доступ до таблиць, не час їх виконання.

Рекомендується використовувати ролі моніторингу, щоб надати підвищені привілеї ролям, без необхідності надавати їх ролі супер-користувача та користуванню всією гнучкістю способів моніторингу. При використанні такого підходу потрібно переконатися, що інші ролі мають лише необхідні для їх роботи привілеї. Під час налаштування баз даних, а особливо таблиць в ній, та доступу до них , потрібно звернути увагу на ролі користувачів та доступи до даних, які вони повинні мати та не повинні.

Є такий спосіб налаштування контролю доступу який називається списком контролю доступу (список керування доступом). Списки контролю доступу це насамперед додаткові рядки які під'єднуються до рядків, функцій, представлень та таблиць, як ми розглядали раніше. Будь-яка грамотно розроблена система повинна поєднувати ролі та список керування доступом для захисту даних. Рекомендується щоб таблиці та інші взаємопов'язані об'єкти належали ролі, що не являється супер-користувачем. Потрібно створити групові ролі, які будуть спроможні відображати дозволи або ролі, котрі мають необхідні привілеї бази даних. Не рекомендується надавати ці права користувачу, що використовується кінцевими користувачами, оскільки даний підхід може достатньо ускладнити можливість гнучкого керування.

Є потреба в зменшенні привілеїв лише до необхідних тож потрібно використовувати підхід групових ролей для спрощення керування. Встановимо застосування шифрування. Існують обставини коли є необхідність шифрувати конфіденційні дані при зберіганні. За допомогою спільноти є необхідна кількість розширень для впровадження додаткової безпеки за допомогою шифрування. Одним з таких є pgcrypto. Це стандартне розширення PostgreSQL, у якого метою є розширення базового інструментарію SQL для надавання функцій шифрування та хешування, котрі потрібно використовувати для моделювання внутрішньої логіки в базі даних. Після встановлення необхідних пакунків на стороні сервера вистачає виконання команди від ролі супер- користувача:

«*CREATE EXTENSION pgcrypto;*». При наявності потреби використання шифрування даних потрібно уважно визначити в яких місцях його потрібно, щоб відповідати нормативним та подібним вимогам. Найпростіший спосіб шифрування є шифрування симетричний. Цей спосіб шифрування не вимагає PGP (програма що дозволяє виконувати роботу шифрування та дешифрування), саме тому можливо її використовувати з базового скачування та налаштування. Для прикладу ми можемо викликати прості SQL команди що зможуть зашифрувати або розшифрувати дані:

«*SELECT pgp_sym_decrypt(pgp_sym_encrypt('Привіт', 'any_key'), 'any_key ');*»

Як ми бачимо в прикладі «pgp_sym_decrypt» функція дешифрує дані, а функція «pgp_sym_encrypt» їх шифрує за допомогою кодового ключа. Але більш безпечним способом шифрування, та його рекомендується використовувати, є асиметричний метод. Він базується на закритому та відкритому ключах які перед використанням потрібно згенерувати. Після успішного згенерування цих ключів є можливість використовувати наступні команди SQL:

«*pgp_pub_encrypt('<дані>', '<відкритий ключ>')*»;

«*pgp_pub_decrypt(<текст шифру>, '<приватний ключ>')*».

Перед вибору методу шифрування потрібно уважно ознайомитися з програмою та даними які там використовуються задля визначення потрібного вам методу шифрування. Це є дуже важливою частиною оскільки деякі дані повинні бути в неявному вигляді задля безпеки. Також варто взяти до уваги, що відкритий ключ зазвичай має більше сенсу під час обміну даними з іншими (оскільки спільного секрету немає), тоді як симетричний може мати більше сенсу для автономної програми.

Що стосується хешування. Хешування необоротнє, тож потрібно відрізняти місця де потрібно використати шифрування, а де хешування. Іншими словами можливо сказати що при хешуванні даних ми отримуємо їх контрольну суму в вигляді символів та можемо це значення використовувати щоб побачити чи були дані змінені або чи надав користувач таке саме значення. Хешування досить використовуєма річ тож її частіше використовують для конфіденційних даних, чутливих даних, та паролів. Їх можливо потрібно перевірити однак не

повертати. Однак важливо пам’ятати що якщо увімкнений та налаштований аудит та використовуються SQL команда що зберігає різного роду чутливу інформацію то воно може бути записана в журнали аудиту. На такий випадок можливо потрібно більш коректно налаштувати аудит, налаштувати мережевий зв'язок на шифрування чи хешувати дані на стороні сервера до виклику SQL команд. Також рекомендуємо не зберігати чутливі дані (паролі, номери карток та інше) користувачів у вигляді простого тексту або відносної (туманної) форми в базі даних.

Іноді бувають випадки коли цього вимагає система, наприклад менеджер зберігання паролів, однак тоді потрібно попіклуватися про додатковий шар безпечних заходів задля дотримання безпеки. Потрібно використовувати одностороннє хешування у всіх місцях системи для перевірки актуальності даних де їх не потрібно повертати. Завершальним етапом рекомендацій є налаштування керування ключами. Достатньо часто розробники та архітектори системи мають проблему пов’язану з шифруванням даними, а з керування ключами. В звичайній формі ваша програма має єдиний або декілька централізованих постійних ключів, які вона використовує під час шифрування та дешифрування даних. В такому випадку є ймовірність що ключ змінити не вийде, тож даний ключ або їх деяка кількість буде використовуватись як користувачами так і вашою програмою на протязі її життя. На такий випадок можливо припустити що даний ключ будуть мати певна кількість людей котра не буде на постійній основі працювати в компанії. На такий випадок на допомогу вам може прийти системи керування ключами.

Не обов’язково рекомендується використовувати підхід системи керування ключами, однак це являється непоганою практикою щодо їх обслуговування. Дані системи полегшують контроль проблеми що була описано вище, а саме дають змогу зберігати ключі в окремій захищеній службі, яка розташована як окремий сервіс від бази даних та зв’язаних з нею програм. Але саме цілю даного сервісу є незалежний контроль ключів для різних користувачів. Візьміть до уваги такий підхід для керування криптографічними ключами, задля відокремлення їх зберігання від програми або уникнення застосування спільних ключів.

Висновок

У статті розглянуто рекомендаційні методи забезпечення безпеки СУБД PostgreSQL на сьогоднішній день для сучасного підприємства, але розглянуті технології забезпечення безпеки потрібно впроваджувати згідно з специфікою їх функціонування.

Перелік посилань

1. Налаштування захисту PostgreSQL від автоматизованих хакерських атак. [Електронний ресурс] – Режим доступу: https://www.8host.com/blog/zashhitapostgresql-ot-avtomatizirovannyx-xakerskix-atak/.
2. Фільтр Postgres: реалізація термінації та моніторингу Postgres SSL. [Електронний ресурс] – Режим доступу: https://github.com/envoyproxy/envoy/issues/10942.
3. Баргилевич Олександр Анатолійович. Дослідження способів і протоколів аутентифікації в інформаційно-телекомунікаційні системах. Всеукраїнська наукова конференція «Актуальні проблеми кібербезпеки». Державний Університет Телекомунікацій, 27 жовтня 2021. Тези доповідей. С. 61 - 62. [Електронний ресурс] – Режим доступу: http://www.dut.edu.ua/uploads/p_2099_79407917.pdf.
4. Internet Security Glossary, Version 2. [Електронний ресурс] – Режим доступу: https://www.rfc-editor.org/rfc/rfc4949.html.
5. Налаштування безпеки сервера PostgreSQL під управлінням Ubuntu. [Електронний ресурс] – Режим доступу: https://4ybakut2004.medium.com
6. Managing PostgreSQL users and roles. [Електронний ресурс] – Режим доступу: https://aws.amazon.com/ru/blogs/database/managing-postgresql-users-androles/.

Надійшла: 18.08.2022 Рецензент: д.т.н., професор Гайдур Г.І.