Министерство образования и науки РФ
ФГБОУ ВПО «Томский государственный университет
систем управления и радиоэлектроники»
Кафедра комплексной информационной безопасности
электронно-вычислительных систем (КИБЭВС)

К.С. Сарин, А.С. Киселёв

БЕЗОПАСНОСТЬ СИСТЕМ БАЗ ДАННЫХ

Лабораторный практикум
для студентов специальностей и направлений
10.03.01 – «Информационная безопасность»,
10.05.02 – «Информационная безопасность
телекоммуникационных систем»,
10.05.03 – «Информационная безопасность
автоматизированных систем»,
10.05.04 – «Информационно-аналитические системы безопасности»

В-Спектр
Томск, 2020

УДК 004.6
ББК 32.972.134
С 20

С 20 **Сарин К.С., Киселев А.С.** Безопасность систем баз данных: лабораторный практикум. – Томск: В-Спектр, 2020. – 44 с.
ISBN 978-5-91191-421-9

Практикум содержит описания лабораторных работ по дисциплине «Безопасность систем баз данных» для специальностей 10.05.02 – «Информационная безопасность телекоммуникационных систем», 10.05.03 – «Информационная безопасность автоматизированных систем», 10.05.04 – «Информационно-аналитические системы безопасности» и направления 10.03.01 – «Информационная безопасность», задания, методические указания по выполнению, требования по представлению отчётности, вопросы для самоконтроля.

УДК 004.6
ББК 32.972.134

ISBN 978-5-91191-421-9

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Лабораторный практикум подготовлен с целью обучения студентов специальностей 10.05.02 – «Информационная безопасность телекоммуникационных систем», 10.05.03 – «Информационная безопасность автоматизированных систем», 10.05.04 – «Информационно-аналитические системы безопасности» и направления 10.03.01 – «Информационная безопасность» работе с базовыми механизмами администрирования операционных систем. Выполнив лабораторные работы, студенты приобретут умения и навыки защиты данных в системах управления базами данных и освоят ряд профессиональных компетенций по обеспечению безопасности систем баз данных.

ЛАБОРАТОРНАЯ РАБОТА № 1

Шифрование

Цель работы

Цель работы – изучение основ шифрования, его целей и задач, а также практическая реализация шифрования в СУБД MS SQL Server.

Теоретические сведения

Шифрование – такое преобразование данных, которое делает прочтение этих данных невозможным без обратного преобразования – расшифрования. Преобразования выполняются на основе некоторого алгоритма шифрования с использованием секрета – ключа шифрования. Используется шифрование для поддержания конфиденциальности данных – прочтение данных становится невозможным посторонним лицам.

Существуют два вида схем шифрования: симметричные (используется один ключ для шифрования и для расшифрования) и асимметричные (используется открытый ключ для расшифрования и закрытый ключ для шифрования). Симметричное шифрование чаще всего используется для непосредственного шифрования рабочих данных (отличается более быстрым процессом преобразования). Асимметричное шифрование обычно используется для подтверждения авторства (передача сообщений, электронная подпись, сертификаты) и шифрования ключевой информации.

В рамках MS SQL Server реализовано три вида шифрования: общая схема шифрования, работающее в рамках этой схемы прозрачное шифрование данных (Transparent Data Encryption) и постоянное шифрование (Always Encrypted).

Целью общей схемы шифрования является предоставление разработчикам базы данных возможности зашифровать свою базу так, как это представляется удобным организации и/или совместимым с местным законодательством по шифрованию.

В рамках общей схемы предоставляются симметричное и асимметричное шифрование, а также сертификаты (можно как использовать внешний сертификат, так и создать собственный). Шифрование по общей схеме реализуется согласно приведённому колесу иерархии шифрования (рис. 1.1). Шифровать по общей схеме шифрования можно конкретные столбцы данных.

В рамках общей схемы шифрования существуют следующие виды механизмов шифрования:

– Transact-SQL: простейшее шифрование на основе парольной фразы с помощью 128-мибитного алгоритма Triple DES. Команды: ENCRYPTBYPASSPHRASE для шифрования и DECRYPTBYPASSPHRASE для расшифрования. Предназначен данный вид шифрования для преобразования строки данных (строка будет сохранена в зашифрованном виде).

– Сертификат: подписанное электронной подписью утверждение о принадлежности открытого ключа шифрования определённому лицу, устройству, сервису и т.д., у которого находится соответствующий закрытый ключ расшифрования. Сертификаты содержат в себе публичный ключ субъекта, его идентификационную информацию, электронную подпись субъекта и имеют заданный срок действия. Сертификаты бывают подписанные доверенным сервисом и самоподписанные. В рамках SQL Server возможно использование внешних сертификатов и создание самоподписанных. Предназначен сертификат для создания доверенного соединения и используется, как правило, для получения необходимых секретных данных от доверенного сервера.

– Асимметричный ключ: пара открытый-закрытый ключи, используемая для зашифровывания секретной информации. Из-за медленной скорости выполнения асимметричных алгоритмов рекомендуется зашифровывать только ограниченный объём информации. Как правило, асимметричные ключи используются для шифрования другой секретной информации. В рамках SQL Server сертификат выполняет схожую с асимметричным ключом роль: они оба используют асимметричную схему шифрования и не имеют различий в надёжности алгоритма шифрования. Различие в том, что к сертификату прилагается электронная подпись, на основании которой возможно создание доверенного соединения.

– Симметричный ключ: один ключ, используемый как для шифрования, так и для расшифрования. Предназначен для шифрования непосредственной информации, алгоритмы симметричного шифрования достаточно быстры для обработки больших объёмов информации.

– Прозрачное шифрование данных: подробно описано далее.

– Обработка внешних ключей (Enterprise Key Management, EKM): схема, используемая для обработки внешних ключей (симметричных и асимметричных), предоставляемых внешними приложениями.

Рис. 1.1. Колесо иерархии шифрования

Прозрачное шифрование данных используется для зашифровывания всей базы данных наиболее удобным способом: для выполнения такого шифрования достаточно исполнить несколько запросов в Transact SQL. Данные в TDE шифруются симметрично, ключ шифрования защищается сертификатом либо асимметричным ключом, шифруется вся база данных вместе с журналами и мета данными в реальном времени. Возможны два варианта зашифровывания с помощью TDE: с помощью мастер-ключа базы данных или с помощью внешнего ключа (оба варианта показаны на рис. 1.1 пунктирными линиями). Данные, так же, как и в общей схеме, шифруются на носителе и расшифровываются при чтении.

Важно отметить, что создаваемые в рамках SQL Server криптографические объекты (прежде всего это ассиметричные ключи и сертификаты) предназначены для использования только в рамках самого SQL Server. Microsoft не рекомендует создавать эти криптографические данные для использования в других приложениях или операционной системой.

Постоянное шифрование предназначено для шифрования данных на ином уровне, нежели общая схема, и существует оно отдельно от общей схемы. В каком-то смысле постоянное шифрование является улучшением общей схемы, поскольку закрываются те недостатки, которые имели место в рамках общей схемы.

В рамках постоянного шифрования шифруются столбцы, данные зашифровываются вплоть до момента непосредственного чтения пользователем с особой привилегией (без таковой пользователь будет видеть только шифротекст) и установленным на клиенте приложением, поддерживающим Always Encrypted (например, SQL Server Management Studio, но могут быть и пользовательские приложения) и имеющим доступ к хранилищу, в котором расположен необходимый ключ. Такие ключи могут располагаться локально на клиентской машине (с помощью хранилища Windows Certificate Store) или удалённо на сервере (с помощью хранилища Azure Key Vault).

В рамках самой базы данных хранятся два вида ключей (оба зашифрованы): мастер-ключ столбца (Column Master Key) и ключ шифрования столбца (Column Encryption Key). Мастер-ключ столбца используется для зашифровывания одного или нескольких ключей шифрования столбца, а ключ шифрования столбца используется для шифрования одного конкретного столбца.

Возможно шифрование определённое (одинаковые данные шифруются одинаковым шифротекстом) или случайное (одинаковые данные шифруются различным шифротекстом с помощью добавления случайного элемента); с определённым шифрованием возможны некоторые операции (структурирование, сортировка, точечный поиск, индексация), однако есть возможность вывести содержимое этих столбцов, если имеется достаточно внешней ин-

формации и/или вариантов содержимого столбца немного (например, да/нет, муж/жен и пр.).

Используется данный вид шифрования для засекречивания отдельных важных данных, которые следует раскрыть только весьма ограниченному кругу лиц (например, номер кредитной карты): такие данные не могут напрямую просматривать высоко привилегированные пользователи, а также к ним не удастся получить доступ злоумышленнику в том случае, если ему удастся взломать сервер базы данных, потому что ключевая информация, необходимая для расшифровывания, хранится не на нём (либо локально на клиенте, либо на специальном сервере).

На стороне сервера базы данных реализовывается постоянное шифрование с помощью графического интерфейса, на стороне клиента всё зависит от реализующей программы (в рамках SQL Server Management Studio необходимо в строке подключения указать параметр подключения column encryption setting=enabled и иметь доступ к хранилищу, содержащему необходимую ключевую информацию).

Ход работы

В ходе работы будет решена задача соблюдения конфиденциальности данных с помощью шифрования конфиденциальной информации.

Практически в любой базе данных хранится информация, оставлять открытой которую организации было бы как минимум нежелательно. Простейший пример такой информации – номера кредитных карт клиентов, с которыми организация работает. Для сокрытия такой информации от посторонних лиц используется разделение прав доступа к объектам базы данных. Однако эта мера работает только на уровне системы управления базами данных, а чтобы хранимые данные не было возможности считать посторонним пользователям в обход системы, их необходимо преобразовать (зашифровать) в форму, которая понятна только при обратном преобразовании (расшифровывании), и данные, используемые при данном преобразовании, хранить во внешних источниках или самой СУБД.

Рассматриваемая задача распределяется на следующие шаги:

1) определение информации в базе данных, подлежащей зашифровыванию;

2) определение степени конфиденциальности таковой информации (на основе как важности самой информации и требованиям к скорости работы с ней, так и среды, в рамках которой данная информация обрабатывается);

3) определение режима и параметров шифрования согласно строгости требований к поддержанию конфиденциальности;

4) применение указанного режима шифрования к информации.

Описание рассматриваемого примера базы данных

Перед тем, как перейти к выполнению поставленной задачи, необходимо знать, с какой базой данных предстоит работать.

Компания Adventure Works Bicycles, Inc. является вымышленным оптовым продавцом велосипедов. Она производит и продаёт велосипеды, а также продаёт одежду и аксессуары к велосипедам розничным торговцам по всей стране.

Сотрудники компании выполняют типичные деловые операции, такие как:

- продажи;
- обработка и выполнение заказов;
- управление запасами;
- управление персоналом;
- бюджетирование.

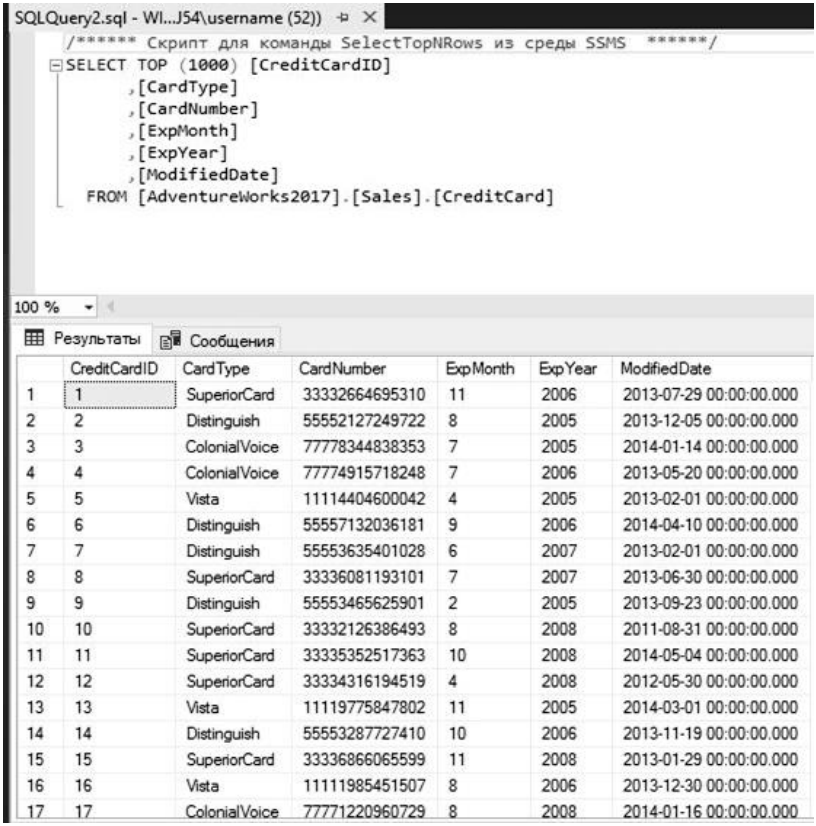
Таблицы в базе данных подразделяются по категориям (схемам):

- dbo – системные базы данных, предназначенные для администрирования;
- HumanResources – информация о сотрудниках компании;
- Person – обобщённые таблицы для описания человека (сотрудника, покупателя, поставщика);
- Production – описание продаваемых компанией товаров;
- Purchasing – приобретаемые у поставщиков детали для велосипедов и прочие товары;
- Sales – информация о продажах, продавцах и покупателях.

Полноценное описание всего содержимого рассматриваемой базы данных дано в документе по следующему адресу: https://dataedo.com/download/AdventureWorks.pdf.

Шаг 1

Откройте базу данных AdventureWorks2017 и найдите в ней таблицу Sales.CreditCard (таблица с информацией по кредитным картам покупателей компании). Ознакомьтесь со структурой данной таблицы (рис. 1.2).



	CreditCardID	CardType	CardNumber	ExpMonth	ExpYear	ModifiedDate
1	1	SuperiorCard	33332664695310	11	2006	2013-07-29 00:00:00.000
2	2	Distinguish	55552127249722	8	2005	2013-12-05 00:00:00.000
3	3	ColonialVoice	77778344838353	7	2005	2014-01-14 00:00:00.000
4	4	ColonialVoice	77774915718248	7	2006	2013-05-20 00:00:00.000
5	5	Vista	11114404600042	4	2005	2013-02-01 00:00:00.000
6	6	Distinguish	55557132036181	9	2006	2014-04-10 00:00:00.000
7	7	Distinguish	55553635401028	6	2007	2013-02-01 00:00:00.000
8	8	SuperiorCard	33336081193101	7	2007	2013-06-30 00:00:00.000
9	9	Distinguish	55553465625901	2	2005	2013-09-23 00:00:00.000
10	10	SuperiorCard	33332126386493	8	2008	2011-08-31 00:00:00.000
11	11	SuperiorCard	33335352517363	10	2008	2014-05-04 00:00:00.000
12	12	SuperiorCard	33334316194519	4	2008	2012-05-30 00:00:00.000
13	13	Vista	11119775847802	11	2005	2014-03-01 00:00:00.000
14	14	Distinguish	55553287727410	10	2006	2013-11-19 00:00:00.000
15	15	SuperiorCard	33336866065599	11	2008	2013-01-29 00:00:00.000
16	16	Vista	11111985451507	8	2006	2013-12-30 00:00:00.000
17	17	ColonialVoice	77771220960729	8	2008	2014-01-16 00:00:00.000

Рис. 1.2. Таблица Sales.CreditCard

Шаг 2

Доступ к базе данных в рамках организации может быть произведён удалённо, в том числе по незащищённому каналу связи. В рамках данной работы рассматриваются два случая:

1) вся база данных расположена на одном сервере;

2) база данных распределена по различным серверам.

Таблица, приведённая на шаге 1, содержит информацию о типе кредитных карт покупателей, дате её окончания и о их номерах – данная информация является конфиденциальной и должна быть максимально защищена вне зависимости от внешних обстоятельств.

В первом случае необходимо шифрование базы данных по частям, зашифровывая только значимую информацию, потому что один сервер был

составлен без значительных запасных производственных ресурсов и не сможет справиться с шифрованием всей базы данных.

Во втором случае возможно шифрование всей базы данных, потому что каждый из множества серверов содержит в себе достаточный набор производственных ресурсов для шифрования своей части базы данных.

Шаг 3

В первом случае необходимо шифрование отдельных таблиц базы данных. Необходимо использование общей схемы шифрования и отбор отдельных таблиц, подлежащих зашифровыванию.

Во втором случае допустимо использование схемы прозрачного шифрования данных (Transparent Data Encryption) для зашифровывания всей базы данных.

Что касательно критической информации (таблица Sales.CreditCard, выбранная на шаге 1), в обоих случаях данные о кредитных картах покупателей критически уязвимы, и их необходимо защитить от максимально большего числа угроз, включая ренегатство со стороны администраторов баз данных. Для данных о кредитных картах целесообразно использовать механизм постоянного шифрования (Always Encrypted).

Шаг 4

Часть 1. TDE

В рамках TDE зашифровывается целиком вся база данных. Шифрование базируется либо на внешнем ключе (с помощью EKM), либо с использованием сертификата от сервера. В рамках данной лабораторной работы рассматривается использование сертификата.

Для выполнения TDE создайте новый запрос, нажав на соответствующую кнопку на панели инструментов, расположенной в верхней части окна SSMS. Введите следующий запрос (в качестве пароля, вводимого на третьей строке, используйте случайную строку символов длиной от 8; в качестве поясняющей информации к сертификату укажите свою фамилию):

```
USE master; -- Используем схему «мастер» – действие происходит на уровне мета данных сервера
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<UseStrongPasswordHere>';      -- Создаётся мастер-ключ базы данных на основе пароля
GO
CREATE CERTIFICATE MyServerCert WITH SUBJECT = 'My DEK Certificate';
-- Создаётся сертификат, подтверждающий ключевую информацию
GO
USE AdventureWorks2017;   -- Используем схему конкретной базы данных – действие происходит на уровне самих данных
GO
```

```
CREATE DATABASE ENCRYPTION KEY  -- Создаём ключ шифрования базы данных (не обозначен в колесе иерархий шифрования, т.к. существует только в рамках TDE); данный ключ является симметричным.
WITH ALGORITHM = AES_128  -- Определяем алгоритм шифрования
ENCRYPTION BY SERVER CERTIFICATE MyServerCert;  -- Определяем подтверждающий сертификат
GO
ALTER DATABASE AdventureWorks2017  -- Изменяем базу данных
SET ENCRYPTION ON;  -- Включаем режим шифрования согласно заданным параметрам (с помощью ключа DEK, созданного ранее)
GO
```

В рамках данного запроса последовательно выполняется следующее:

I. На уровне мета данных сервера (системной базы данных master):

1) создаётся мастер-ключ базы данных (DMK), защищающийся с помощью пароля;

2) создаётся сертификат.

II. На уровне непосредственной базы данных:

3) создаётся симметричный ключ шифрования базы данных (Database Encryption Key) – ключ, используемый только в рамках TDE для шифрования всей базы данных. При создании ключа определяется алгоритм шифрования и сертификат, зашифровывающий этот ключ;

4) необходимая база данных зашифровывается с помощью этого ключа.

Возможно также использование асимметричного ключа вместо сертификата для шифрования DEK. Для этого, соответственно, необходимо заменить код создания сертификата на код создания асимметричного ключа, и также при создании DEK указать, каков будет защищающий его ключ. Создание асимметричного ключа возможно с помощью конструкции CREATE ASYMMETRIC KEY. Ключ может быть либо создан с нуля, либо экспортирован из внешнего источника (файла или поставщика ключей). По умолчанию ключ, как и сертификат, зашифровывается с помощью мастер-ключа базы данных, но может быть также зашифрован и паролем. Простейший пример конструкции, которая создаст с нуля пару асимметричных ключей на основе алгоритма RSA с размером 2048 бит и зашифрует их мастер-ключом базы данных:

```
CREATE ASYMMETRIC KEY MyAssymKey
WITH ALGORITHM = RSA_2048;
```

Прозрачное шифрование данных является, собственно, прозрачным. Со стороны пользователя узнать о его существовании не только сложно, но и нет в этом никакой нужды. Чтобы проверить, запущен ли режим шифрова-

ния, повторите команду включения режима шифрования базы данных (ALTER DATABASE AdventureWorks2017 SET ENCRYPTION ON; в рамках базы данных: USE AdventureWorks2017;) и убедитесь в том, что выполнение запроса возвращает ошибку, в которой утверждается о том, что шифрование уже запущено (рис. 1.3).

Часть 2. Always Encrypted

В качестве примера рассмотрим столбец CardNumber в рамках рассмотренной ранее таблицы Sales.CreditCard.

Always Encrypted реализуется с использованием мастера («визарда»). Откройте мастер, выбрав в контекстном меню (таблицы или столбца) пункт «Зашифровать столбцы». В первую очередь в мастере выбирается столбец и параметры шифрования (рис. 1.4).

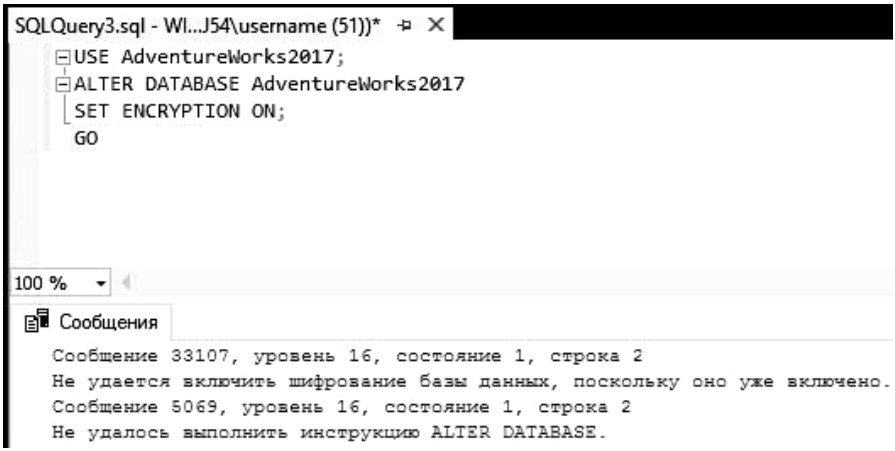


Рис. 1.3. Шифрование базы данных включено

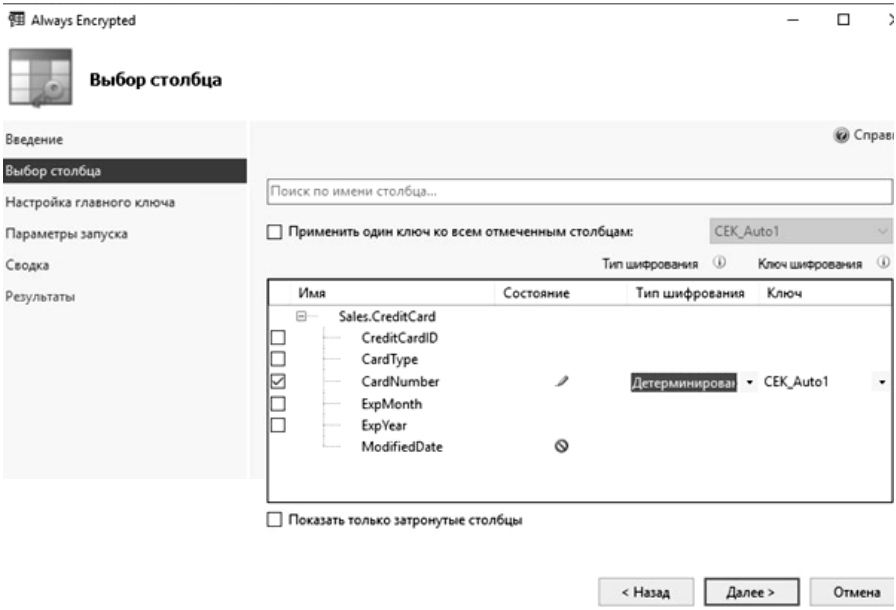


Рис. 1.4. Выбор столбца

На данной вкладке определяются зашифровываемые столбцы, тип шифрования и ключ шифрования (имеющийся либо новый).

Если был использован новый ключ шифрования, необходимо указать, к какому мастер-ключу он имеет отношение (рис. 1.5) и каков источник этого ключа. Так как главный ключ ещё не был создан, возможен выбор только автоматического создания. Поскольку в рамках данной лабораторной работы рассматривается одна машина, в качестве поставщика хранилища оставьте хранилище сертификатов Windows.

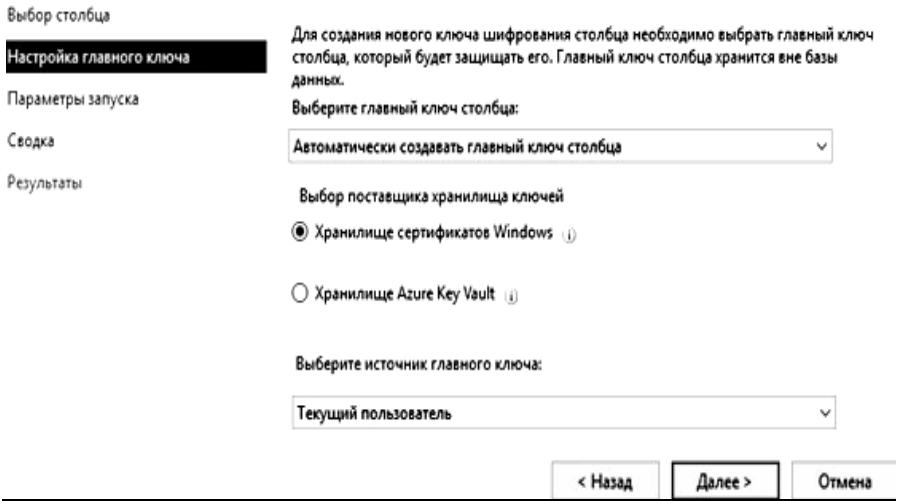


Рис. 1.5. Настройка главного ключа

На вкладке «Параметры запуска» определяется, когда будет запущен скрипт шифрования. В рамках реального предприятия возможно отложить процедуру шифрования до момента наименьшей нагрузки на сервер, в рамках же лабораторной работы оставьте немедленное исполнение.

На вкладке «Сводка» приводится для ознакомления итоговый набор параметров, собранный мастером.

После завершения работы мастера становится видно, что указанный столбец (столбцы) зашифрован (рис. 1.6).

Рис. 1.6. Зашифрованный столбец

В таком виде данные хранятся в базе данных, и так их будут видеть все, кроме тех, кто подключится к базе данных с особым параметром и с использованием соответствующих ключей из хранилища.

Проверим считывание зашифрованных данных. Для этого необходимо вновь подключиться к СУБД, при этом необходимо указать в дополнительных параметрах параметр column encryption setting=enabled (рис. 1.7).

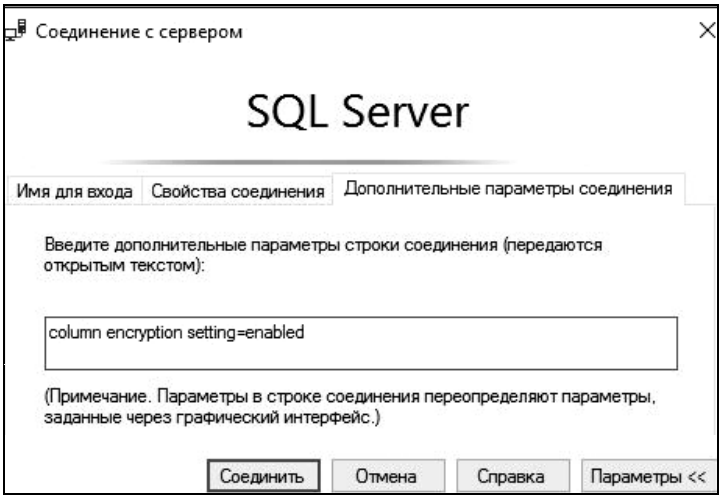


Рис. 1.7. Указание параметра соединения

Далее выполните запрос на чтение к таблице, содержащей зашифрованную информацию (в контекстном меню таблицы «Выбрать первые 1000 строк»). SSMS сообщит о том, что для обработки данных, находящихся в зашифрованном столбце, необходимо включить определение параметров (рис. 1.8). Подтвердите включение. Убедитесь, что данные успешно расшифрованы.

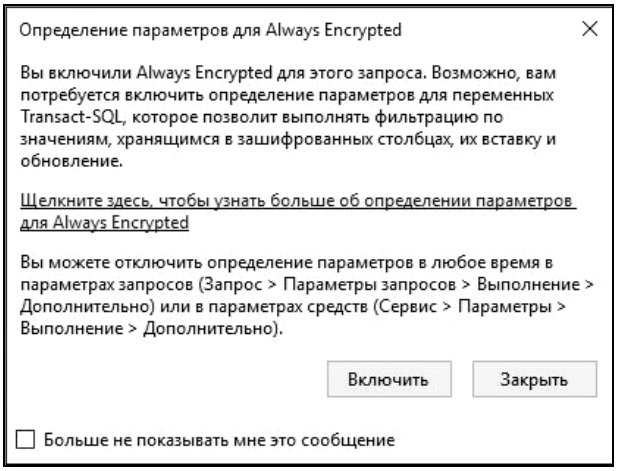


Рис. 1.8. Определение параметров для Always Encrypted

В определённый момент – например, при изменении структуры таблицы – может возникнуть необходимость выключить режим шифрования AE для каких-либо столбцов. Выполняется отмена режима с помощью того же мастера, с помощью которого шифрование включалось. Необходимо на вкладке «Выбор столбца» у зашифрованного столбца указать в пункте «Тип шифрования» – «Открытый текст» (рис. 1.9). Выполните процедуру отмены режима шифрования.

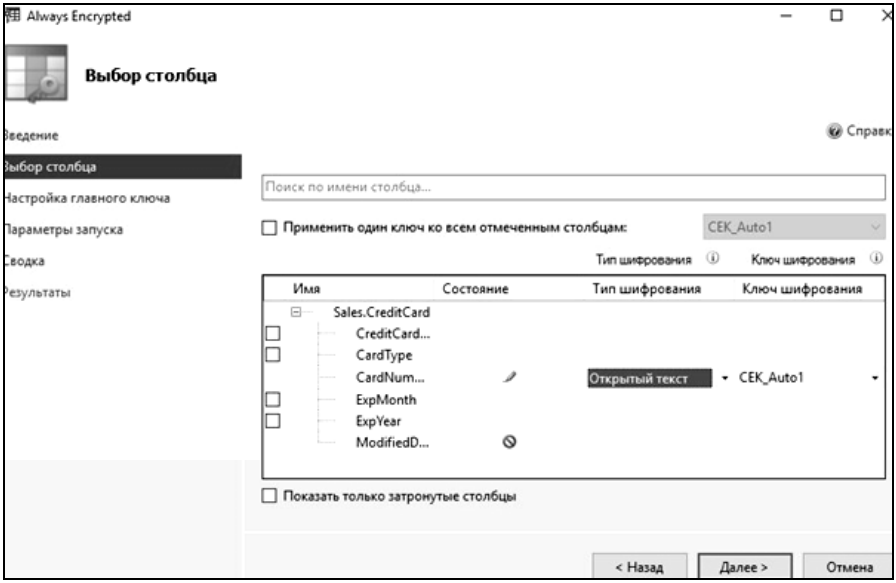


Рис. 1.9. Расшифровывание столбца

Ключ расшифрования ключевой информации (главный ключ столбца) хранится либо удалённо в хранилище Azure, либо локально в хранилище сертификатов Windows, что определяется при включении режима шифрования. Локально размещённый ключ можно взять из хранилища сертификатов и перенести в другую машину. Чтобы открыть это хранилище, введите в окне запуска (WIN+R) certmgr.msc. Ключ будет храниться в папке Личное и будет назван Always Encrypted Auto Certificate (рис.1.10).

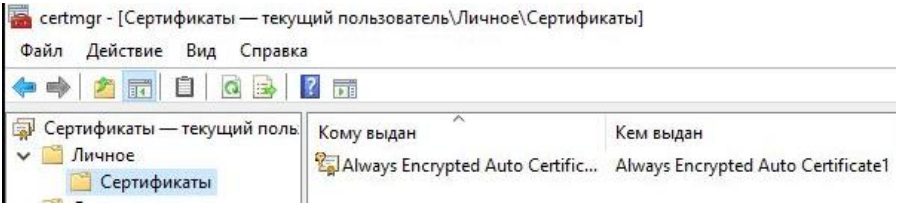


Рис. 1.10. Ключ Always Encrypted в хранилище сертификатов Windows

Контрольные вопросы

1) Что такое шифрование и как оно реализовывается?
2) Опишите симметричную и асимметричную схему шифрования. Для чего используется открытый ключ, для чего – закрытый?

3) Какие есть механизмы шифрования в рамках общей схемы шифрования SQL Server?

4) Опишите связь между сертификатом и асимметричным ключом и направления их использования.

5) Опишите процесс шифрования по прозрачному шифрованию данных (TDE).

6) Для чего предназначены и как связаны мастер-ключи (сервисный и базы данных) (SMK и DMK)?

7) Что такое и для чего предназначено постоянное шифрование (Always Encrypted)?

8) Какими ключами оперирует постоянное шифрование и для чего эти ключи предназначены?

Задание

Выполните Ход работы согласно вариантам. Составьте отчёт по выполненной работе. Защитите отчёт.

Задание по TDE (прозрачному шифрованию данных): зашифруйте базу данных AdventureWorks2017 согласно варианту.

Таблица 1.1 – Варианты заданий по TDE

№	Алгоритм шифрования DEK	Защита DEK с помощью
1	AES_256	сертификата
2	AES_256	асимметричного ключа
3	AES_128	сертификата
4	AES_256	асимметричного ключа
5	AES_192	сертификата
6	AES_256	асимметричного ключа
7	AES_128	сертификата
8	AES_128	асимметричного ключа
9	AES_192	сертификата
10	AES_192	асимметричного ключа

Задание по AE (постоянному шифрованию данных): ознакомьтесь с описанной ситуацией и зашифруйте столбец с помощью режима AE согласно

ней. В качестве режима шифрования используйте определённый. Варианты заданий:

1) При написании обзоров на товары компании Adventure Works покупатели оставляли своё имя и электронную почту. Контактные данные обзорщиков необходимо скрывать от всеобщего доступа и защищать.

2) Велосипеды компании не так просты, как кажутся. В их составе множество различных мелких деталей. Некоторые из них изготавливаются по секретным разработкам компании, добавляя их продукции особенности, которые и придают компании конкурентоспособности. Не стоило бы про них знать конкурентам.

3) Согласно ФЗ «О персональных данных», персональными данными являются «любые сведения, относящиеся к прямо или косвенно определённому или определяемому физическому лицу (субъекту персональных данных), которые предоставляются другому физическому или юридическому лицу либо лицам». Пожалуй, более определяющих данных, нежели номер паспорта, у физического лица быть не может. Такие данные, естественно, подлежат защите. Правда, в рамках компании, поскольку она расположена в Америке, используется несколько другой номер – social security number, который является частным случаем более общего понятия – national identification number.

4) Каждый сотрудник имеет право хранить размер своей заработной платы в секрете. Бездушной автоматизированной машине, обрабатывающей цифры его зарплаты, эта информация нужна. Вполне возможно, она также нужна и кассиру. Но в открытом доступе она точно не нужна (ПРИМЕЧАНИЕ: для этого варианта необходимо перед шифрованием удалить ограничение на столбец).

5) На основании того, что человек приобретает, легко определить его интересы. Этим могут воспользоваться не только легитимные организации по желанию самих пользователей, но и спамеры. Поскольку данные самих пользователей защищаются в других таблицах, необходимо скрыть сопоставление пользователя с покупателем.

6) Несмотря на то, что пароли пользователей хранятся в хешированном виде, их всё равно необходимо хранить в зашифрованной форме, потому что по большому количеству хешей злоумышленник может почерпнуть полезную для себя информацию, например, определить функцию хеширования.

7) От того, сколько продавец продаёт товаров, зависит в том числе и его оплата. Соответственно, такая информация должна быть конфиденциальна. Однако данные о самих продажах нужны многим отделам, в том числе они выкладываются в открытый доступ в рамках ежегодного отчёта в сово-

купной форме. Таким образом, необходимо скрыть продажи конкретных продавцов.

8) Никто не хочет обрабатывать «выгодные предложения, от которых просто невозможно отказаться», к тому же на своём рабочем телефоне, который нельзя просто так отключить или проигнорировать. Или телефоне, который был дан покупателем компании и которая обязалась его, как и все прочие пользовательские данные, защищать. Некрасиво получится, если номера таких телефонов будут в открытом доступе.

9) По каждому продукту организации имеется минимум одна техническая документация. В списке таких документов имеются в том числе и документы ДСП, например, инструкция по производству определённых деталей. Поскольку в составе продукции компании имеются собственные засекреченные разработки, такие документы следовало бы закрыть от общего доступа.

10) Далеко не все пользователи, разбрасывающие свои электронные почтовые адреса направо и налево, готовы к спаму. Компания, конечно, уважает своих пользователей и рассылает им только действительно важную информацию. Но будут ли настолько добры злоумышленники, продающие спамерам украденные почтовые адреса? Сомнительно.

Дополнительное задание: покажите на примере своего варианта, или любого другого шифрования в рамках SQL Server, что до шифрования данные возможно считать из файла базы данных, а после – невозможно. При этом не разрешается использовать никакие прикладные к СУБД программы-интерфейсы (SQL Server Management Studio является таковой).

ЛАБОРАТОРНАЯ РАБОТА № 2

Разделение прав доступа

Цель работы

Целью работы является изучение принципов работы разделения прав доступа в базах данных на примере СУБД Microsoft SQL Server.

Теоретический материал

Разделение прав доступа – способ разграничения субъектам доступа к объектам базы данных.

Обычный пользователь в SQL Server имеет имя входа и имя пользователя. По имени входа пользователь осуществляет вход в систему, имя пользователя определяется на уровне всего экземпляра SQL Server. Имя пользователя определяется для каждой базы данных отдельно, на основе имени пользователя осуществляется непосредственное разграничение доступа к данным. Один пользователь может иметь имя пользователя в нескольких базах данных одновременно, но в каждой отдельной базе данных у пользователя имеется только одно имя пользователя. Аналогично именам на уровне сервера пользователи могут быть членами роли сервера, и на уровне базы данных пользователи могут быть членами роли базы данных.

Виды имён входа разделяются по тому, каким образом осуществляется вход: аутентификация SQL Server, Windows или Active Directory.

При создании имени пользователя определяется, каким образом (по какому имени входа) у него осуществляется доступ к базе данных. Имя пользователя может быть создано и без имени входа; пользователь, естественно, не может зайти в базу данных под этим именем, но ему можно выдавать разрешения.

Список объектов, к которым может иметь доступ субъект, довольно обширен. По областям объекты могут принадлежать уровню сервера, базы данных или схемы. На уровне сервера объекты общие (роль сервера, имя входа, база данных и пр.), на уровне базы данных объекты общие для всей базы данных (криптографические данные, служба, пользователь, роль приложения, схема и пр.), на уровне схемы объекты наиболее детализированные (тип схемы, таблица, представление, процедура, внешняя таблица и пр.).

В SQL Server имеется набор предопределённых ролей и имён. На уровне сервера имеется набор ролей для различного администраторов:

– sysadmin – полный доступ;

– serveradmin – параметры конфигурации и выключение сервера;

– securityadmin – распределение прав доступа, сброс паролей (может назначать разрешения, поэтому почти эквивалентен sysadmin);

– processadmin – завершение процессов в сервере;

– setupadmin – добавление и удаление связанных серверов (при работе в SSMS необходимо также быть в роли sysadmin);

– bulkadmin – инструкция BULK INSERT (импорт базы данных или таблицы в виде файла);

– diskadmin – управление файлами;

– dbcreator – создание, удаление, изменение, восстановление баз данных.

Также имеется роль public – наиболее общая роль, которая не предоставляет особенных привилегий и к которой в обязательном порядке принадлежат все имена входа. Предназначена она для выдачи прав доступа, которые

должны иметь все пользователи. Изменять права доступа ролей сервера можно только для роли public и пользовательских ролей.

Имеется одно предопределённое имя входа: sa. Пользователь sa (системный администратор) имеет доступ ко всем субъектам и ни в чём не ограничен, однако само имя входа можно отключить. При установке экземпляра SQL Server создаётся пользователь с аутентификацией Windows, также наделённый всеми правами, под которым был создан этот экземпляр.

На уровне базы данных также имеется роль public. Эта роль по умолчанию имеет множество прав доступа, которые необходимы для самых базовых процессов в базе данных и не предоставляют угрозы безопасности. Права доступа этой роли также можно менять. Все имена входа по умолчанию принадлежат к этой роли. Имеются также и некоторые другие предопределённые роли, название которых, как правило, говорит само за себя (db_owner – полный доступ, db_securityadmin – разделение прав доступа, db_datareader – полное чтение всех пользовательских таблиц, db_denydatareader – полный запрет на чтение всех пользовательских таблиц, и пр.).

Имеются следующие предопределённые имена пользователя в каждой базе данных: dbo, guest, sys и INFORMATION_SCHEMA. Пользователь dbo имеет доступ ко всем объектам в базе данных и не может быть в них ограничен. Под этим именем подключается набор пользователей, в число которых входят различные администраторы, пользователи роли sysadmin и имени входа sa и владелец базы данных. Пользователь guest (гость) имеет набор разрешений, которые наследуются пользователями, имеющими доступ (CONNECT) к базе данных. Пользователи sys и INFORMATION_SCHEMA являются системными и операции с ними недоступны. Всех этих пользователей нельзя удалить (хотя у guest можно отменить разрешение на CONNECT, что не позволит под ним зайти). Также, при использовании некоторых сертификатов, для их использования создаются системные пользователи, имя которых обрамлено хэш-символами (##).

Для создания имени входа используется инструкция CREATE LOGIN. При создании в основном определяются имя пользователя и пароль, могут также задаваться и параметры вроде необходимости смены пароля при первом входе, ограниченном времени пароля, применения стандартных политик и пр. Чтобы создать имя входа с авторизацией в SQL Server, следует указать имя пользователя и пароль:

CREATE LOGIN <имя> WITH PASSWORD = '<пароль >';

а чтобы создать имя входа с авторизацией в Windows, следует указать домен и имя пользователя:

CREATE LOGIN [<домен>\<имя>] FROM WINDOWS;

Для создания имени пользователя используется инструкция CREATE USER. Чаще всего создаются пользователи с привязкой к имени входа, для чего оно указывается:

CREATE USER <имя пользователя> FOR LOGIN <имя входа>;

Иногда, впрочем, необходимо создать пользователя без входа, для чего вместо FOR LOGIN <имя входа> указывается WITHOUT LOGIN. Это используется, как правило, для выдачи разрешений перед предоставлением доступа. Чтобы впоследствии сопоставить такого пользователя с именем входа, используется инструкция ALTER USER <имя пользователя> с указанием нужного имени входа (WITH LOGIN <имя входа>). Также не имеют возможности входа пользователи, созданные на основе сертификатов и асимметричных ключей. Такие пользователи создаются для подписи соответствующих криптографических модулей.

Чтобы создать и удалить роль, используются инструкции CREATE и DROP. Их синтаксис таков:

CREATE SERVER ROLE <имя роли> '<имя входа владельца роли>' -- создание роли уровня сервера (если не указано имя входа владельца, им станет исполнитель инструкции)

DROP SERVER ROLE <имя роли> -- удаление роли уровня сервера (возможно только если роль не имеет членов)

CREATE ROLE <имя роли> '<имя пользователя владельца роли>'-- создание роли уровня базы данных (при исполнении инструкции убедитесь, что находитесь в рамках нужной базы данных, что определяется в окне SSMS на самой нижней из верхних панели слева)

DROP ROLE <имя роли> -- удаление роли уровня базы данных (аналогично)

Чтобы сопоставить имя с ролью, необходимо использовать инструкцию ALTER. Синтаксис:

ALTER ROLE <имя роли> ADD MEMBER <имя пользователя>; --включение имени пользователя в роль базы данных

ALTER SERVER ROLE <имя роли> ADD MEMBER <имя входа>; --включение имени входа в роль сервера

ALTER ROLE <имя роли> DROP MEMBER <имя пользователя>; --удаление имени пользователя из роли базы данных

ALTER SERVER ROLE <имя роли> DROP MEMBER <имя входа>; --удаление имени входа из роли сервера

Также этой инструкцией можно изменить имя роли:

ALTER (SERVER) ROLE <имя роли> WITH NAME = <новое имя>;

Также многие объекты из описанных (роли, имена входа, имена пользователей) можно создать и настроить с помощью графического интерфейса SSMS. Происходит это в рамках папки «Безопасность», расположенную в корне для объектов уровня сервера и в корне базы данных для объектов уровня базы данных (Рисунок 2.1).

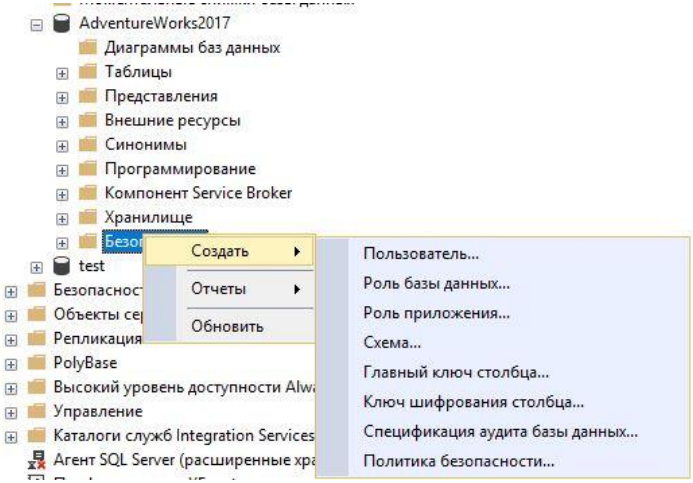


Рисунок 2.1 – Создание субъектов безопасности

С каждым объектом в SQL Server связан набор разрешений, которые могут быть выданы субъектам. Всего в SQL Server имеется 230 видов разрешений (многие из них специфицированы для определённых объектов). Разрешения предоставляются и отнимаются командами GRANT, REVOKE и DENY. GRANT выдаёт разрешение, REVOKE отнимает выданное напрямую данному субъекту разрешение, DENY запрещает этому субъекту выполнение указанного права вне зависимости от прочих разрешений. Невозможно изменить разрешения системным именам (sys, sa, INFORMATION_SCHEMA и dbo), владельцу сущности и пользователю, исполняющему процесс изменения прав доступа (самому себе).

Основной список разрешений, с которым осуществляется работа при разделении прав доступа, это разрешения пользователям на SELECT, INSERT, UPDATE и DELETE определённых таблиц, представлений и столбцов в них. Также следует отметить следующие разрешения: CONTROL, дающее полный доступ к объекту; EXECUTE, позволяющее запускать функции; ALTER (ANY), позволяющее изменять конкретный объект (все объекты указанного типа); CONNECT, позволяющее осуществлять подключение (отмена этого разрешения вообще запретит пользователю подключаться, то есть практически отберёт все права без их отъёма на уровне самой СУБД); и CREATE, позволяющее создавать объекты указанного типа.

Синтаксис этих команд специфичен для различных видов объектов. Простейший вариант выглядит следующим образом:

GRANT [список разрешений] ON [объект] TO [субъект];
DENY [список разрешений] ON [объект] TO [субъект];

REVOKE [список разрешений ON [объект] FROM [субъект];

Обработка прав доступа с ролями аналогична обработке прав доступа с именами.

Также для некоторых объектов (в том числе таблиц) возможна настройка прав доступа в графическом интерфейсе SSMS. Для этого следует открыть свойства нужного объекта, вкладку «Разрешения» и с помощью кнопки «Найти...» определить нужный субъект, которому далее указать необходимые права (Рисунок 2.2). В рамках таблицы, для некоторых разрешений, где это применимо, возможно указание прав доступа к конкретным столбцам (Рисунок 2.3).

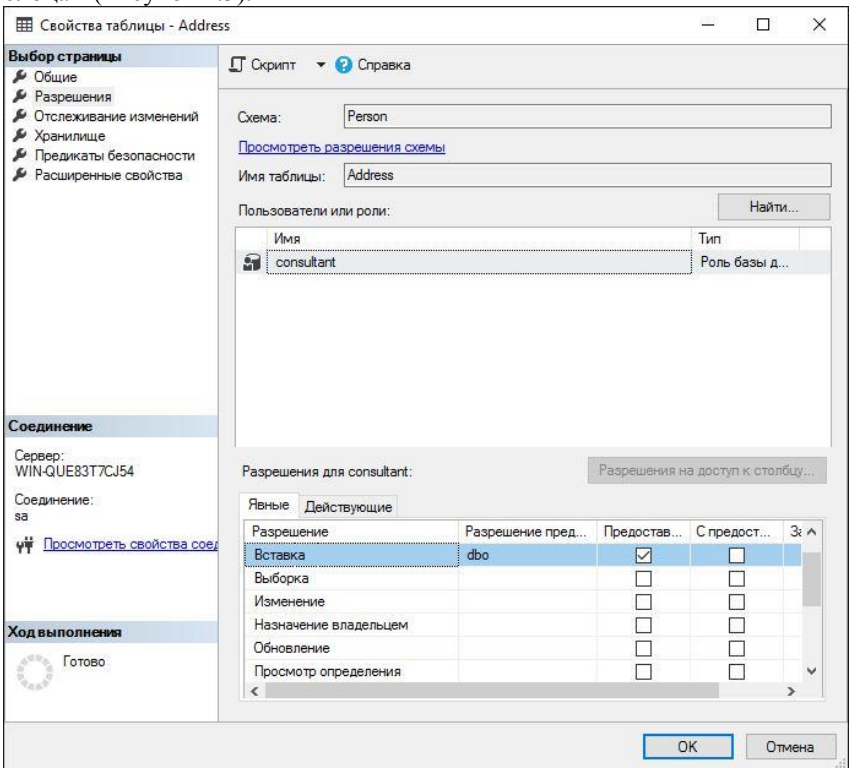


Рисунок 2.2 – Задание прав доступа субъекту

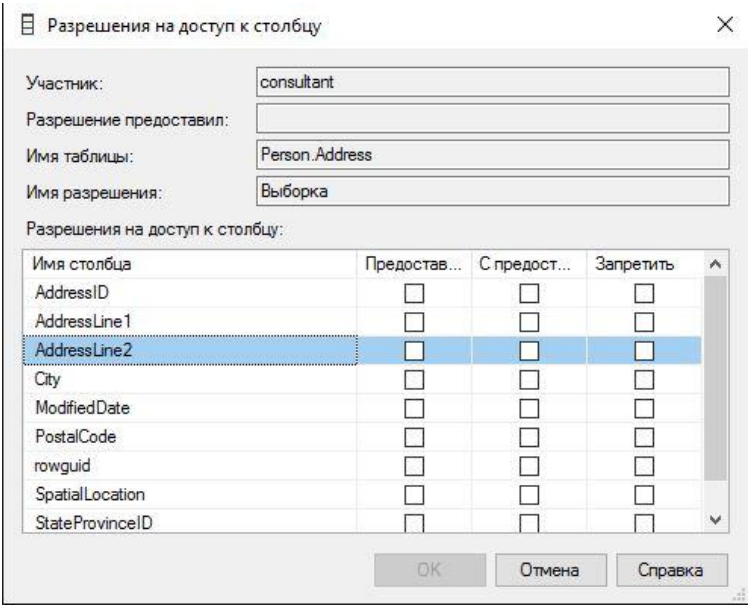


Рисунок 2.3 – Задание прав доступа субъекту на уровне столбцов

Ход работы

В ходе работы будет выполнена задача соблюдения конфиденциальности данных за счёт разделения прав доступа.

Наиболее основным средством соблюдения конфиденциальности является разделение прав доступа. Этот инструмент буквально выполняет определение конфиденциальности, а именно разрешает доступ тем, кто должен его иметь, и запрещает всем остальным. Система разделения прав доступа в SQL Server весьма гибка. В её рамках возможно разделение прав на самых различных уровнях: от возможности создавать базы данных и настраивать безопасность сервера до доступа к конкретным строкам (защита на уровне строк рассмотрена в следующей лабораторной). На практике, чем крупнее организация, тем более мелкие и точные выдаются права доступа каждому сотруднику, тем больше выполняется правило минимальных прав доступа. И в рамках SQL Server возможно достаточно точное разделение прав за счёт наличия 230 различных видов разрешений на множество субъектов, от полного доступа до возможности выполнения конкретной операции. Однако всё-таки основным защищаемым объектом является сама информация, то есть содержимое таблиц в базах данных.

Рассматриваемая задача разделяется на следующие шаги (в рамках одной пользовательской БД):

1) определение списка всех пользователей, которые должны работать с базой данных, и их прав доступа (к бизнес-сущностям);

2) определение списка ролей согласно всем уникальным случаям разделения прав;

3) создание (или сопоставление с предопределёнными) необходимых ролей и наделение их требуемыми правами доступа;

4) создание пользователей и сопоставление их с ролями.

Шаг 1

Для примера рассмотрим одного пользователя: консультанта. Консультант – это специалист по работе с клиентами, задачами которого являются демонстрация клиенту товара, помощь в выборе товара, сбор отзывов от клиентов. Соответственно, консультанту необходим доступ к таблицам, в которым содержится базовая информация о товаре, его местонахождении, стоимости, а также документация к товару.

Шаг 2

В рамках нашего примера используется только один пользователь, поэтому роль будет только одна. На уровне сервера консультанту особых привилегий не требуется, поэтому он будет членом только роли public. На уровне базы данных создадим новую роль и назовём эту роль consultant.

Шаг 3

Для создания роли можно воспользоваться следующей инструкцией:

```
CREATE ROLE consultant;
```

Либо же воспользоваться графическим интерфейсом, выбрав в контекстном меню любой из папок: «Безопасность», «Роли», «Роли базы данных» пункт «Создать» -> «Роль базы данных», и в появившемся окне указать имя роли (Рисунок 2.4).

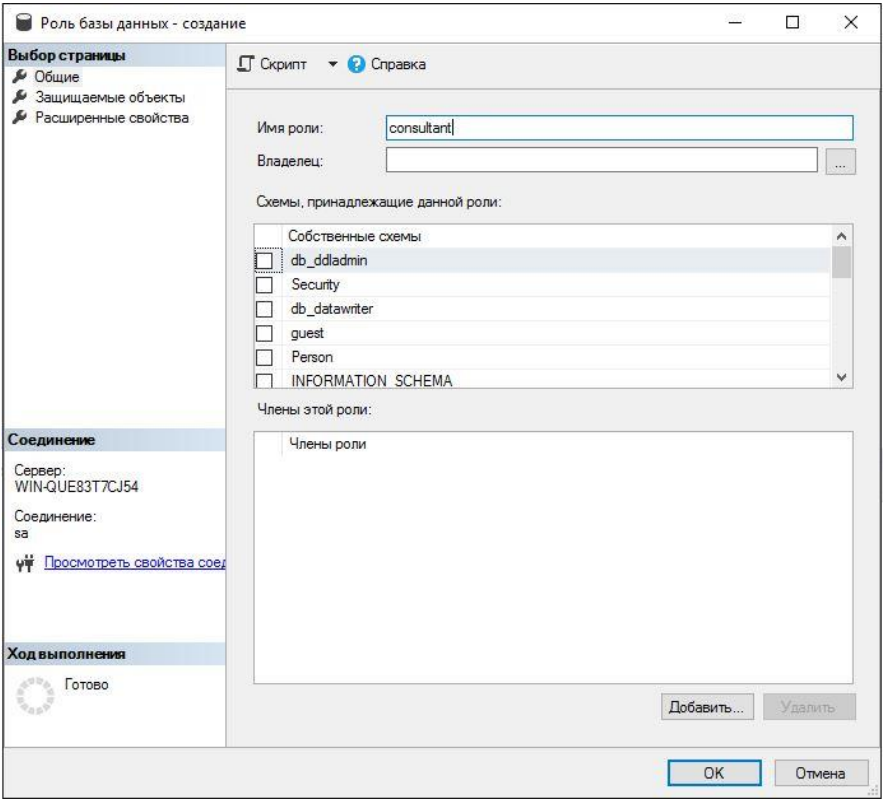


Рисунок 2.4 – Создание роли базы данных

Владельцем роли по умолчанию станет её создатель, то есть, скорее всего (если вы зашли в СУБД под именем входа sa или через аутентификацию Windows), пользователь dbo.

С учётом необходимых прав, определённых на первом шаге, консультанту нужны будут следующие таблицы со следующими правами (Таблица 2.1):

Таблица 2.1 – Определение необходимых прав доступа

Права/таблица	Production.Product	Production.Document	Production.ProductCategory	Production.ProductDescription	Production.ProductPhoto	Production.ProductReview

SELECT	+	+	+	+	+	+
INSERT	-	-	-	-	-	+
UPDATE	-	-	-	-	-	+
DELETE	-	-	-	-	-	+

Таблицы необходимы ему только на чтение, кроме таблицы с оформлением обзора от клиента. Клиент может обозревать продукт в магазине (без онлайн доступа), поэтому консультант будет вносить данные в базу за него. Клиент также может изменить свой обзор и отказаться от него, поэтому права будут на все базовые действия.

Далее необходимо наделить роль правами. Сделать это возможно как с помощью графического интерфейса (в свойствах каждой таблицы), так и с помощью команд. Покажем команду на примере таблицы с обзорами пользователей:

GRANT SELECT, INSERT, UPDATE, DELETE ON Production.ProductReview TO consultant;

Наделите роль консультанта заданными правами.

Шаг 4

Когда все права определены, можно создавать пользователей и давать им доступ к базе данных. Создадим имя входа с аутентификацией SQL Server. Для создания имени входа воспользуйтесь инструкцией:

CREATE LOGIN username WITH PASSWORD = '12345';

Если выдаётся ошибка о слишком простом пароле, добавьте к паролю заглавные и строчные английские буквы и знаки препинания, а также учтите минимальную длину в 8 символов.

Пользователя также можно создать с помощью графического интерфейса. Для этого, из корневой папки «Безопасность», выберите «Создать» -> «Имя входа» и определите нужные параметры имени входа (Рисунок 2.5).

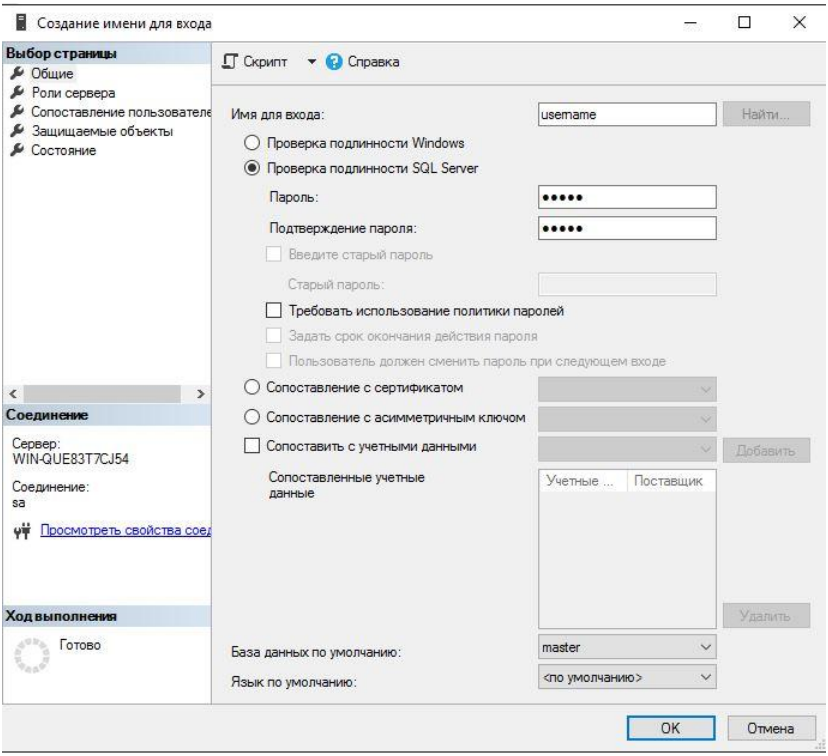


Рисунок 2.5 – Создание имени входа

Далее необходимо создать имя пользователя в рамках базы данных AdventureWorks2017. Делается это аналогично созданию имени входа, также возможно создание с помощью графического интерфейса (Рисунок 2.6) или TRANSACT-SQL. Инструкция (убедитесь, что находитесь на уровне базы данных AdventureWorks2017):

CREATE USER username FOR LOGIN username;

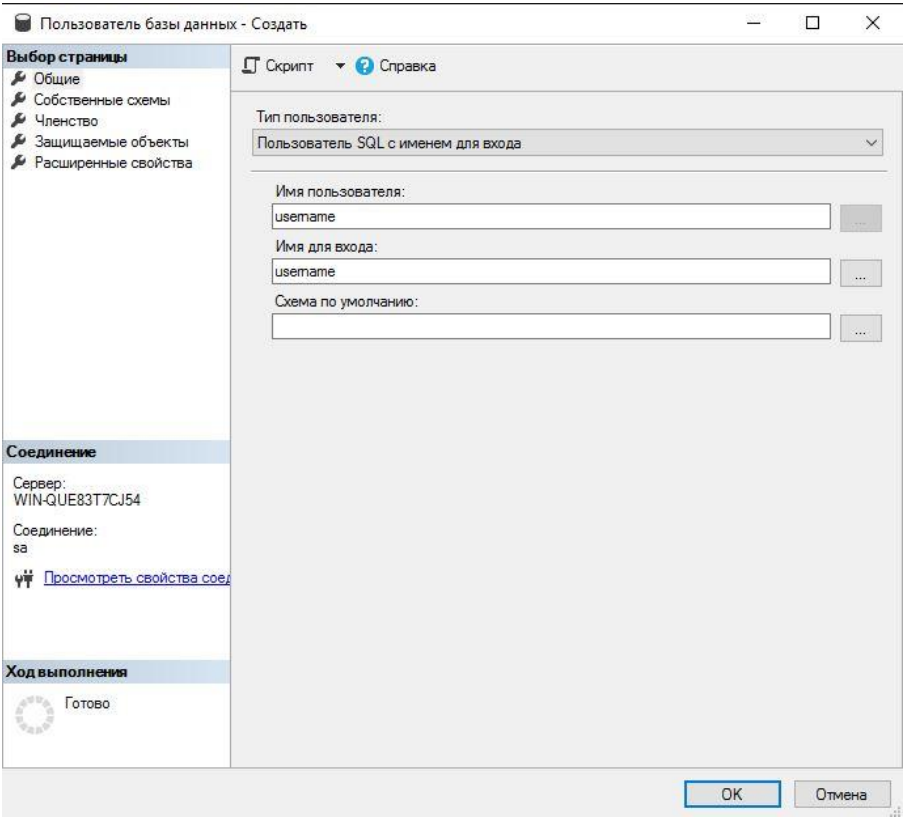


Рисунок 2.6 – Создание имени пользователя

Контрольные вопросы

1) Опишите систему имён и ролей в SQL Server;
2) Назовите несколько предопределённых ролей сервера и опишите их права;
3) Назовите несколько предопределённых ролей базы данных и опишите их права;
4) Назовите все предопределённые имена входа и имена пользователя и опишите их права;
5) Как пользователь, не имеющий никаких явно выделенных ему прав в рамках базы данных и не имеющий явного членства в ролях, может осуществлять простейшие операции, такие как подключение к базе данных, чтение списка баз данных и пр.?

6) Какими командами создаются, изменяются и удаляются имена и роли? Какими командами выдаются, отзываются и запрещаются права?
7) Опишите различие между отзывом права и запретом на право.

Задание

Первое задание: создайте имя входа и присвойте ему предопределённую роль сервера согласно указанным в варианте правам (Таблица 2.2). Создайте другое имя входа и имя пользователя в рамках базы данных AdventureWorks2017, сопоставьте их и присвойте имени пользователя предопределённую роль согласно указанным в варианте правам (список ролей можно посмотреть в папке «Безопасность» -> «Роли» -> «Роли базы данных»).

Таблица 2.2 – Варианты к первому заданию

Номер варианта	Права роли сервера	Права роли БД
1	Доступ без ограничений	Строгий запрет на чтение
2	Работа с файлами	Выполнение команд DDL
3	Настройка и отключение сервера	Работа с данными
4	Работа с базами данных	Управление правами на удалённый доступ
5	Доступ без ограничений	Строгий запрет на изменение
6	Работа с правами доступа	Чтение данных
7	Настройка и отключение сервера	Управление разрешениями
8	Инструкция BULK INSERT	Владелец базы данных
9	Работа с базами данных	Создание резервной копии
10	Остановка процессов	Выполнение команд DDL

Второе задание: на основании заданной в варианте профессии создайте роль уровня базы данных и пару имя входа/имя пользователя, определите круг данных, к которым должен выделяться доступ, и на основании этого круга выберите минимум 5 таблиц (либо обоснуйте, почему меньшего числа достаточно) и права доступа (рассмотрите только 4 права: SELECT, INSERT, UPDATE, DELETE) к ним и выдайте определённые права доступа созданной роли. Составьте таблицу согласно определённым правам (аналогично Таблице 2.1). Вкратце (1-2 предложения на таблицу) объясните выбор таблиц и прав доступа к ней. Сопоставьте имя пользователя с ролью. Продемонстрируйте работу преподавателю.

Внимание: не опускайтесь уровня ниже таблицы. Со строками работа будет осуществляться в рамках следующей лабораторной работы.

1) бухгалтер;
2) ведущий собеседование по найму на работу;
3) сотрудник отдела кадров;
4) продавец;
5) покупатель на сайте организации (незарегистрированный);
6) покупатель на сайте организации (зарегистрированный, права отдельно от незарегистрированного);
7) дизайнер онлайн магазина;
8) разработчик новой продукции;
9) кассир (выдаёт зарплату сотрудникам);
10) производитель продукции.

Дополнительное задание: в рамках своего варианта, согласно выполненному второму заданию, рассмотрите все столбцы выбранных вами таблиц. Уточните права к каждому из столбцов. Обоснуйте свой выбор (вкратце для каждого столбца).

ЛАБОРАТОРНАЯ РАБОТА № 3

Защита на уровне строк

Цель работы

Целью работы является изучение принципов работы защиты на уровне строк (Row-Level Security) на примере СУБД Microsoft SQL Server.

Теоретический материал

Защита на уровне строк – способ разграничения прав доступа к содержимому таблицы на уровне строк (разграничение права идёт построчно) в зависимости от характеристик пользователя, который выполняет запрос, или каких иных условий. Основная причина ввода такого рода разграничений – облегчение процесса программирования базы данных: ранее процесс разграничения прав доступа в рамках одной таблицы (одного набора таблиц) осуществлялся с помощью использования представлений, функций и триггеров (что является ресурсозатратным и трудномасштабируемым), или на основе логики приложения (что является небезопасным, так как злоумышленник может обойти логику приложения и воспользоваться всеми правами, которыми обладает приложение). Защита на уровне строк действует на уровне базы данных и применяется для всех без исключения соединений, что позволяет уменьшить поверхность атаки. Механизм защиты определяется для таблицы.

В SQL Server защита на уровне строк осуществляется на основании предикатов безопасности. Реализуется данная защита с помощью средств TRANSACT-SQL. Существуют два вида предикатов: фильтр и блокировка.

Фильтр скрытно от пользователя (прозрачно) осуществляет фильтрацию строк, оставляя в результате запроса только те строки, которые удовлетворяют условиям фильтра. Блокировка запрещает выполнение определённых операций при выполнении условия, и если пользователь наткнётся на блок, то ему будет возвращена ошибка.

Предназначение фильтра в скрытии данных при их чтении, и затрагивает он, соответственно, все операции с чтением: SELECT, UPDATE, DELETE (нельзя обновлять и удалять скрытые строки). Однако пользователь может так обновить доступную ему строку, что на неё начнут действовать условия фильтра и он потеряет к ней доступ. Предназначение блокировки в запрещении определённых изменений, и затрагивает она, соответственно, все операции с изменениями: INSERT, UPDATE, DELETE.

У блокировки есть два формата: BEFORE и AFTER. AFTER используется с INSERT и UPDATE, BEFORE – с UPDATE и DELETE. Предикаты с AFTER запрещают пользователю добавлять и изменять значения, попадающие под условие (нарушающие) предиката. Предикаты с BEFORE запрещают пользователю обновлять и удалять значения, которые на данный момент попадают под условие предиката. Особенность работы блока AFTER UPDATE: блок позволяет работать с уже нарушающими условие столбцами. Например, блок запрещает выставлять работнику зарплату выше ста тысяч. Такой блок не запретит изменять адрес работника, чья зарплата уже превышает сто тысяч. Разделение работы блока UPDATE на BEFORE и AFTER накладывает некоторые ограничения на доступный функционал: например, запретить с помощью блока пользователю изменять значение столбца на превышающее текущее (учитывает и текущее значение, и будущее) нельзя. Подобный функционал рекомендуется реализовывать по старинке триггерами.

При создании политики безопасности являются отключёнными. Отключённые политики никак не влияют на процедуры запросов к таблицам базы данных. Однако возможно создать только одну пару предикат-операция (например, на таблицу может быть лишь один предикат фильтра и по одному предикату блока на каждую форму блока: AFTER INSERT, AFTER UPDATE, BEFORE UPDATE и BEFORE DELETE), причём не имеет значения, включена ли политика, вызывающая эти предикаты, или отключена. Также невозможно менять связанные с предикатом столбцы базы данных (не затронутые менять разрешено). Предикаты действуют на всех пользователей, на которых заданы действовать, включая владельца базы данных, системного администратора и прочих системных ролей (хотя владелец базы данных всегда может изменить или удалить эти предикаты; что, впрочем, можно аудировать).

Функционально предикаты полностью эквиваленты тому, что доступно при использовании проверок WHERE. Предикаты основываются на исполнении заранее созданной функции. Предикаты могут задействовать внешние таблицы и/или функции при проверке своих условий. При применении предикатов не рекомендуется осуществлять преобразования типов данных, использовать рекурсии и присоединять чрезмерно много внешних таблиц, чтобы обеспечить приемлемую скорость работы предикатов и обойти возможные ошибки (преобразований и бесконечных рекурсий).

Определяется функционал защиты на уровне строк через ключевое слово SECURITY POLICY. Через его CREATE, ALTER и DROP происходит создание, изменение и удаление политики соответственно. SECURITY POLICY определяет, какие предикаты используются в рамках конкретной таблицы. Простейший пример создания защиты на уровне строк:

```
CREATE SECURITY POLICY Security.SalesFilter -- Создаём политику с заданием её названия в схеме Security
```

```
        ADD FILTER PREDICATE Security.fn_securitypredicate(AppUserId) -- Добавляем предикат-фильтр на основании указанной функции, передавая ей название проверяемого столбца AppUserId
            ON dbo.Sales,  -- Определяем таблицу, на которую задаётся предикат
            ADD BLOCK PREDICATE Security.fn_securitypredicate(AppUserId)   -- Добавляем предикат-блок на основании той же функции
             ON dbo.Sales AFTER INSERT -- Указываем таблицу и режим оперирования блока
        ADD BLOCK PREDICATE Security.fn_securitypredicate(AppUserId)
            ON dbo.Sales AFTER UPDATE -- Аналогично для изменения
            WITH (STATE = ON); -- Включаем политику при создании
```

Данный пример подразумевает, помимо существования таблицы Sales и схемы Security, наличие созданной функции fn_securitypredicate в рамках схемы Security, которая принимает на вход столбец и осуществляет проверку на основании содержимого ячеек этого столбца (столбцов). Пример создания такой функции:

```
        CREATE FUNCTION Security.fn_securitypredicate(@AppUserId int)  -- Создаётся функция с указанием названия и столбца с целыми значениями
            RETURNS TABLE  -- Функция возвращает таблицу
            WITH SCHEMABINDING  -- С привязкой схемы: это означает, что для всех запросов, применяемых в рамках фильтра, пользователю не нужно иметь права доступа к таковым запросам
        AS
            RETURN SELECT 1 AS fn_securitypredicate_result  -- возвращает 1 для каждой строки
            WHERE  -- Строки фильтруются следующим образом:
            DATABASE_PRINCIPAL_ID() = DATABASE_PRINCIPAL_ID('AppUser')    -- ID текущего пользователя на уровне базы данных сравнивается с заданным названием: логика проверки в том, что проверяется совпадение имени текущего пользователя с именем, используемым приложением, и таким образом только через приложение можно получить доступ к данной таблице
            AND CAST(SESSION_CONTEXT(N'UserId') AS int) = @AppUserId;   -- Из сессионного контекста по ключевому слову UserId берётся значение и сравнивается со значениями столбца, название которого определяется при вызове функции; если строка в указанном столбце совпадает со значением контекста (по ключу данного контекста очевидно, что оно будет содержать ID пользователя в приложении), фильтр примет такую строку, и для этой строки будет возвращена единица.
```

Такая функция подразумевает задание сессионного контекста под ключевым словом UserId, иначе фильтр не вернёт ни одной строки. В целом предикат основывает фильтрацию на том, что доступ имеют только пользователи определённого приложения к тем строкам, в которых указан ID этого пользователя в приложении; таковые пользователи не могут просматривать и изменять такие данные (за что отвечает фильтр) и могут вставлять только данные, в которых в столбце имени пользователя будет указано их имя. Блок

имеет смысл только для AFTER, потому что BEFORE уже было убрано из доступа с помощью фильтра.

Если в организации не используется приложение, или же приложение не подразумевает использование сессионного контекста и работает вокруг пользовательской системы самой СУБД, необходимо использовать иные средства определения того, какие строки какой пользователь может просматривать. В таком случае зачастую наиболее подходящим способом будет работа вокруг ролей и сопоставление прав доступа пользователя с его принадлежностью к той или иной роли, а также, если пользовательские данные хранятся в СУБД, возможно сопоставление имени пользователя в базе данных с именем текущего пользователя и фильтрация данных на основании этого сопоставления.

В случае работы с ролями основной функцией будет IS_MEMBER, определяющей членство текущего пользователя в указанной роли, и сопоставление этой роли с данными, соответствующими этой роли. Пример функции для работы вокруг ролей (исходные данные: таблица с данными о студентах, в которой есть столбец с названием факультета):

```
CREATE FUNCTION Security.fn_facultypredicate(@faculty AS sysname)
RETURNS TABLE
WITH SCHEMABINDING
AS
RETURN SELECT 1 AS fn_securitypredicate_result
WHERE
(@faculty == 'РК' AND IS_MEMBER("RK_decanate")) OR – Деканат РК считывает данные только студентов с РК
(@faculty == 'РТ' AND IS_MEMBER('RT_decanate')) OR
(@faculty == 'ФБ' AND IS_MEMBER('FS_decanate'))
```

В случае работы вокруг сопоставления имени пользователя в базе данных с текущим, или иного сопоставления, работающего с именем текущего пользователя, основной функцией будет USER_NAME(), возвращающей имя текущего пользователя (в базе данных) в виде строки данных. Пример функции для работы вокруг имени текущего пользователя (исходные данные: таблица продаж, в которой имеется столбец с именем пользователя):

```
CREATE FUNCTION Security.fn_securitypredicate(@SalesRep AS sysname)
   RETURNS TABLE
WITH SCHEMABINDING
AS
   RETURN SELECT 1 AS fn_securitypredicate_result
```

WHERE @SalesRep = USER_NAME() OR USER_NAME() = 'Manager'; -- продавцы имеют доступ только к своим продажам, а менеджер (пользователь с именем пользователя Manager) может считывать их все

Однако далеко не всегда у разработчиков разделения прав доступа на уровне строк есть возможность изменять структуру таблиц и добавлять туда определяющие столбцы, и им приходится ограничивать права доступа в одной таблице на основании данных, хранящихся в другой. Например (используется БД AdventureWorks):

```
CREATE FUNCTION Security.foo(@bid as sysname)
RETURNS TABLE
WITH SCHEMABINDING
AS
RETURN SELECT 1 AS res
WHERE @bid = (SELECT BusinessEntityID
FROM Person.PersonPhone
WHERE PhoneNumberTypeID = 3 AND BusinessEntityID = @bid);
```

Данная функция предназначена для фильтрации таблицы Person.Person, на основании столбца BusinessEntityID (в качестве переменной @bid), и фильтр по этой функции пропустит только данные о тех людях, у которых тип номера телефона является третьим по идентификатору. В таблице Person.Person при этом нет никаких данных о типе номера телефона, эти данные берутся из таблицы Person.PersonPhone, а сопоставление этих таблиц происходит с помощью встроенного запроса (запроса, находящегося в коде функции в скобках).

Возможно также, что данные понадобится взять из более чем одной внешней таблицы, в таком случае в рамках встроенного запроса эти таблицы объединяются с помощью JOIN на основании общего для них столбца. Например:

```
SELECT BusinessEntityID
FROM HumanResources.Department JOIN HumanResources.EmployeeDepartmentHistory
ON HumanResources.Department.DepartmentID = HumanResources.EmployeeDepartmentHistory.DepartmentID
```

объединяет таблицы HumanResources.Department и HumanResources.EmployeeDepartmentHistory на основании идентификатора отдела (DepartmentID), что позволяет, например, определить, в каком отделе находится сейчас конкретный работник (в таблице HumanResources.EmployeeDepartmentHistory хранятся данные о смене отдела работниками, но их данных об отделе там хранится только внешний ключ – иденти-

фикатор отдела, в то время как в таблице HumanResources.Department эти идентификаторы сопоставляются с параметрами отдела, например, с его названием).

Ход работы

В ходе работы будет выполнена задача соблюдения конфиденциальности данных с помощью разделения прав доступа к данным на уровне строк.

Часто случается так, что к одной и той же таблице базы данных необходимо иметь доступ нескольким лицам, или даже нескольким группам лиц. Однако не каждый из них должен иметь полный доступ ко всей такой таблице. Наиболее простой пример такой таблицы – таблица продаж: сотруднику следует видеть (и обрабатывать) только те продажи, которые были выполнены им самим, но знать о продажах других сотрудников ему не следует. Подобные примеры можно найти в большинстве предметных областей: деканат имеет доступ к данным только тех студентов, которые числятся на его факультете; начальнику отдела необходимы данные только о тех сотрудниках, которые принадлежат его отделу, и пр. Ранее подобный функционал реализовывался с помощью представлений (каждому лицу/роли представлялось своё представление, которое с помощью какого-либо условия обрабатывало одну или несколько исходных таблиц) и функций по внесению изменений в исходные таблицы при изменении этих представлений, а также за счёт триггеров (функций, запускающихся автоматически при выполнении определённых действий, например, запросов к таблицам). Сейчас же есть возможность непосредственного указания условия на доступ к строкам с помощью соответствующего функционала – защиты на уровне строк.

Рассматриваемая задача разделяется на следующие шаги:

1) определение таблиц общего доступа в базе данных;
2) определение круга лиц и/или ролей, которые должны иметь доступ к определённой таблице;
3) определение тех прав, которые должны иметь определённые лица и/или роли;
4) настройка определённых прав с помощью защиты на уровне строк.

Шаг 1

Откройте базу данных AdventureWorks2017 и найдите в ней таблицу Sales.SalesOrderHeader. Ознакомьтесь со структурой данной таблицы (рис. 2.1).

Шаг 2

Обычно для поддержки защиты на уровне строк используется один из трёх методов:

1) создаётся отдельная таблица, сопоставляющая имена пользователей в базе данных с их идентификаторами в базе данных;

2) в рамках необходимой таблицы вводится дополнительный столбец, в котором записывается, какой пользователь имеет право доступа к строке (или какая роль имеет право доступа);

3) процедура реализуется через внешнее приложение, которое передаёт СУБД через сессионный контекст необходимую информацию (идентификатор пользователя), который сопоставляется с данными в таблице.

	Имя столбца	Тип данных	Разрешить ...
▶🔑	SalesOrderID	int	☐
	RevisionNumber	tinyint	☐
	OrderDate	datetime	☐
	DueDate	datetime	☐
	ShipDate	datetime	☑
	Status	tinyint	☐
	OnlineOrderFlag	Flag:bit	☐
	SalesOrderNumber		☐
	PurchaseOrderNumber	OrderNumber:nvar...	☑
	AccountNumber	AccountNumber:n...	☑
	CustomerID	int	☐
	SalesPersonID	int	☑
	TerritoryID	int	☑
	BillToAddressID	int	☐
	ShipToAddressID	int	☐
	ShipMethodID	int	☐
	CreditCardID	int	☑
	CreditCardApprovalCode	varchar(15)	☑
	CurrencyRateID	int	☑
	SubTotal	money	☐
	TaxAmt	money	☐
	Freight	money	☐
	TotalDue		☐
	Comment	nvarchar(128)	☑
	rowguid	uniqueidentifier	☐
	ModifiedDate	datetime	☐
			☐

Рис. 2.1. Таблица Sales.SalesOrderHeader

Возможны и другие варианты, однако указанные являются наиболее простыми и понятными. Чаще всего встречается последний вариант (в подавляющем большинстве организаций роли и имена входа в СУБД используются на уровне администрирования СУБД, а для доступа к самим данным используется один пользователь – имя входа для приложения, в сессионном контексте которого определяется пользователь приложения), поэтому в Ходе работы продемонстрируем именно его.

Создайте имя входа в СУБД: в контекстном меню папки «Безопасность» на уровне соединения выберите «Создать» -> «Вход…». Назовите его app (рис. 2.2). Создайте пользователя в базе данных AdventureWorks2017: в контекстном меню папки «Безопасность» на уровне базы данных выберите «Создать» -> «Пользователь…». Назовите его своими инициалами ФИО и сопоставьте с именем входа app (рис. 2.3). Выдайте пользователю все стандартные права доступа (Выборка, Вставка, Обновление, Удаление, которые соответствуют правам на выполнение запросов SELECT, INSERT, UPDATE, DELETE соответственно) на таблицу Sales.SalesOrderHeader, либо через меню «Свойства» и пункт «Разрешения» нужной таблицы (рис. 2.4), либо через свойства самого пользователя, либо с помощью запроса.

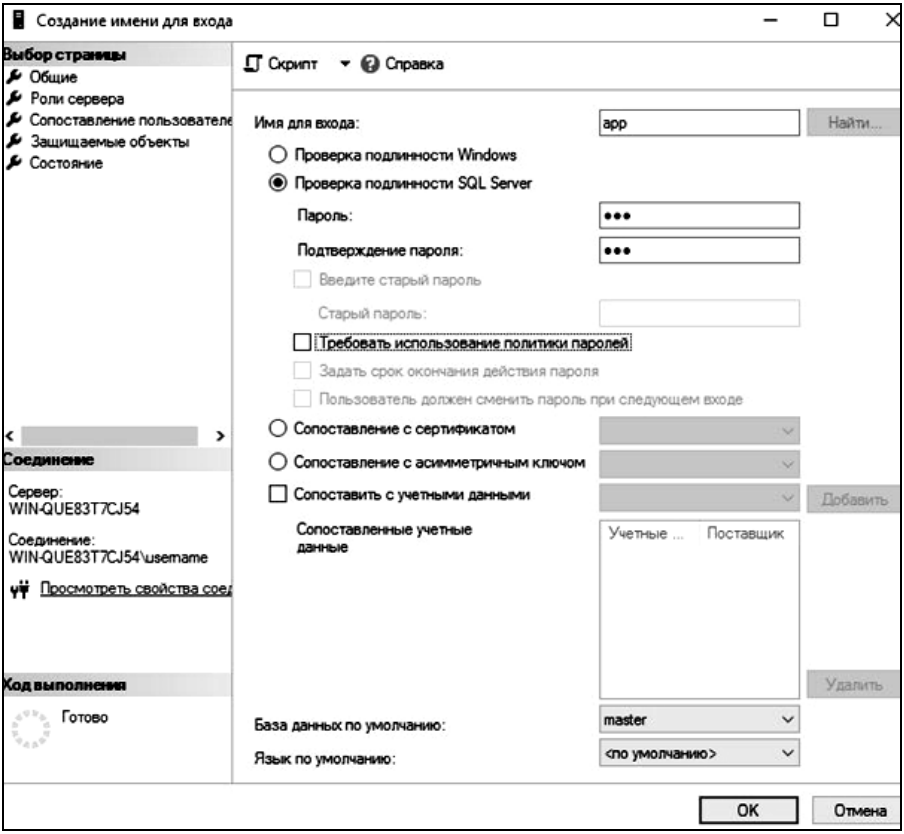


Рис. 2.2. Создание имени входа

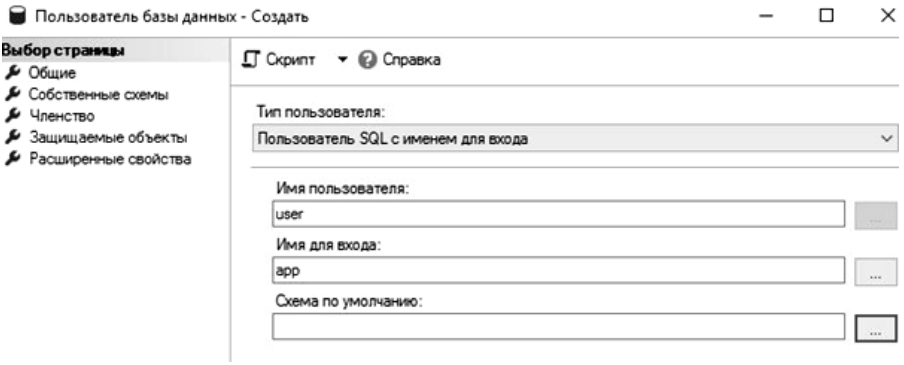


Рис. 2.3. Создание пользователя

С точки зрения СУБД, по построенной нами логике работы, пользователь, использующий внешнее приложение, совпадает с тем пользователем, идентификатор которого приложение представляет в СУБД через сессионный контекст (SESSION_CONTEXT). Его идентификатор является целым числом. Посчитаем, что у этого пользователя идентификатор 275.

Шаг 3

Поскольку был выбран третий вариант, права, которые выдаются пользователям, определяются на основании их идентификатора, который передаётся приложением через сессионный контекст. Предназначением выбранной нами таблицы является сбор всей основной связующей информации по конкретным продажам, с указанием такового идентификатора. Соответственно, разграничением доступа на уровне строк будет фильтрация всех тех строк, идентификатор (SalesPersonID) в которых будет совпадать с идентификатором текущего пользователя, а также запрет на добавление таких строк, или изменение имеющихся строк на такие, или удаление таких строк, которые в столбце SalesPersonID содержат идентификатор, отличный от идентификатора текущего пользователя.

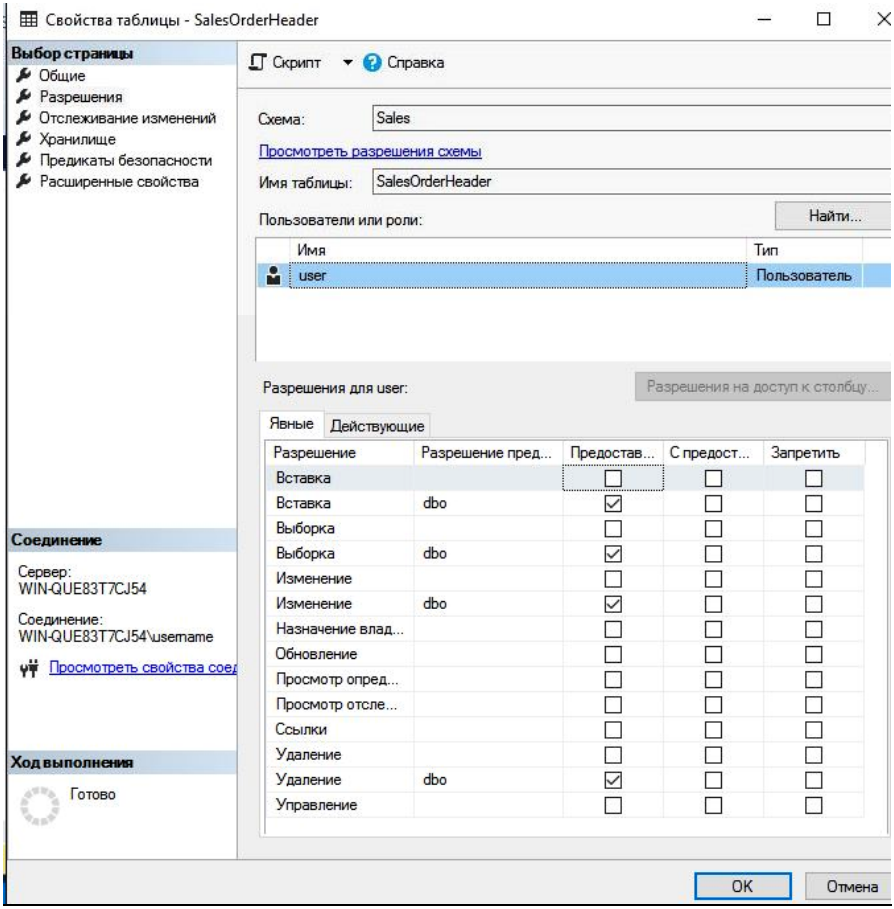


Рис. 2.4. Выдача пользователю прав

Примечание: хотя блок, казалось бы, не имеет значения, так как со стороны приложения подобный функционал должен реализовываться без участия пользователя (выдаваемые пользователю на просмотр таблицы не должны содержать ключевой информации, поскольку в ней у пользователя отсутствует нужда; она нужна для связи таблиц, а таковой функционал вполне реализуем автоматизированными средствами), однако это создаёт дополнительный слой защиты как от возможных случайных ошибок со стороны пользователей и/или приложения, так и со стороны возможного злоупотребления отсутствием таковой проверки злоумышленником.

Шаг 4

Создайте новый запрос в рамках базы данных AdventureWorks2017. Введите следующий код в запрос (на место [ВАШЕ_ФИО] введите имя созданного ранее пользователя):

```
CREATE FUNCTION Security.spred(@SalesPersonId int)  -- Создаётся функция с указанием названия и столбца с целыми значениями в качестве параметра; назвать функцию можно как угодно
    RETURNS TABLE  -- Функция возвращает таблицу
    WITH SCHEMABINDING  -- С привязкой схемы: это означает, что для всех запросов, применяемых в рамках фильтра, пользователю не нужно иметь права доступа к таковым запросам
AS
    RETURN SELECT 1 AS result  -- возвращает 1 для каждой строки; назвать возвращаемое значение можно как угодно
    WHERE  -- Строки фильтруются следующим образом:
    DATABASE_PRINCIPAL_ID() = DATABASE_PRINCIPAL_ID('[ВАШЕ_ФИО]')  -- ID текущего пользователя на уровне базы данных сравнивается с заданным названием: логика проверки в том, что проверяется совпадение имени текущего пользователя с именем, используемым приложением, и таким образом только через приложение можно получить доступ к данной таблице
     AND CAST(SESSION_CONTEXT(N'UserId') AS int) = @SalesPersonId;   -- Из сессионного контекста по ключевому слову UserId берётся значение и сравнивается со значениями столбца, название которого определяется при вызове функции; если строка в указанном столбце совпадает со значением контекста (по ключу данного контекста очевидно, что оно будет содержать ID пользователя в приложении), фильтр примет такую строку.
```

Выполните запрос. Этот запрос создаёт схему Security и в рамках этой схемы определяет функцию безопасности. Создайте и выполните также следующий запрос:

```
CREATE SECURITY POLICY Security.SalesOrderHeader -- Создаём политику с заданием её названия в схеме Security
    ADD FILTER PREDICATE Security.spred(SalesPersonId) -- Добавляем предикат-фильтр на основании указанной функции, передавая ей столбец SalesPersonId из таблицы Sales.SalesOrderHeader
    ON Sales.SalesOrderHeader,  -- Определяем таблицу, на которую задаётся предикат
    ADD BLOCK PREDICATE Security.spred (SalesPersonId) -- Добавляем предикат-блок на основании той же функции
    ON Sales.SalesOrderHeader AFTER INSERT -- Указываем таблицу и режим оперирования блока
ADD BLOCK PREDICATE Security.spred(SalesPersonId)
    ON Sales.SalesOrderHeader AFTER UPDATE -- Аналогично для изменения
    WITH (STATE = ON); -- Включаем политику при создании
```

Этот запрос создаёт и запускает политику безопасности, которая осуществляет фильтрацию и блокировку на основании определённой ранее функции безопасности.

Для проверки того, что права доступа на уровне строк успешно разделены, выполните любые запросы SELECT, INSERT, UPDATE, DELETE от имени входа app (можно либо присоединиться к СУБД в качестве этого имени входа по заданному при создании пароля, либо выполнить запрос от его имени пользователя: EXECUTE AS USER = '[ВАШЕ_ФИО]') с заданием сессионного контекста, равным идентификатору пользователя:

```
EXEC sp_set_session_context @key=N'UserId', @value=275;
```

Убедитесь, что запрос SELECT * FROM Sales.SalesOrderHeader возвращает только те строки, в которых значение столбца SalesPersonID совпадает с указанным в рамках сессионного контекста, и что остальные запросы с участием этого столбца выполняются только в том случае, если значение соответствующего столбца совпадает с идентификатором пользователя. На рис. 2.5 видно, что в результате выполнения операции обновления затронуты не все строки, а только 450 (всего в таблице 31465 строк), то есть только те, в которых SalesPersonID равен числу 275.

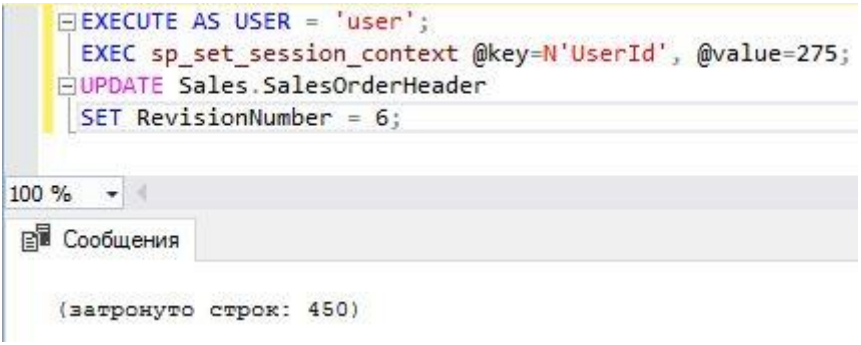


Рис. 2.5. Результат выполнения запроса

Обратите внимание, что если сейчас попробовать взаимодействовать с этой таблицей от имени суперпользователя (под которым был выполнен изначальный вход), то ничего не получится, так как политика безопасности действует на всех пользователей без исключения. Однако пользователи с правами на работу с политиками безопасности могут их включать и отключать, таким образом, за ними остаётся полноценный функционал.

Контрольные вопросы

1) Что есть защита на уровне строк? В чём её предназначение?
2) Какие два вида предикатов защиты на уровне строк существуют в SQL Server? Как они работают?
3) Какие операции затрагивает каждый из видов предикатов?

4) Какие есть форматы у блока? В чём их смысл? С какими операциями они используются?

5) Какая команда включает и отключает предикаты? На чём (каком функционале) основаны предикаты?

Задание

В рамках данной лабораторной работы задание является продолжением задания предыдущей лабораторной работы. Используйте тот же вариант, что и в предыдущей лабораторной работе.

Определите для как минимум одной (наиболее подходящей по смыслу в рамках варианта) определённой во втором задании предыдущей лабораторной работы таблицы разделение прав доступа на уровне строк. Напишите защиту на уровне строк согласно разделению, указанному в варианте этой лабораторной работы. Для задания указанного параметра обязательно воспользуйтесь пользовательским контекстом или иным свойством пользователя (имя пользователя, членство в ролях); защита на уровне строк должна определять разным пользователям (с разным контекстом) различное содержимое таблицы. Указывая различные контексты, убедитесь в работоспособности защиты на уровне строк.

1) учёт данных только сотрудников определённого отдела (параметр: название отдела);
2) работа только с сотрудниками на испытательном сроке (сотрудником на испытательном сроке считается сотрудник, у которого указан код бизнес-сущности, но который при этом является кандидатом);
3) работа в рамках своего отдела (параметр: название отдела);
4) только данные о себе (параметр: код сотрудника);
5) товары только в рамках своего региона (параметр: название региона);
6) только данные о себе (параметр: код покупателя);
7) предложения только своего региона (параметр: название региона);
8) работа с определённым типом велосипедов (параметр: название категории велосипеда);
9) работа в рамках своего отдела (параметр: название отдела);
10) работа с определённым типом велосипедов (параметр: название категории велосипеда).

Список вариантов предыдущей лабораторной работы (для удобства):

1) бухгалтер;
2) ведущий собеседование по найму на работу;
3) сотрудник отдела кадров;

4) продавец;
5) покупатель на сайте организации (незарегистрированный);
6) покупатель на сайте организации (зарегистрированный, права отдельно от незарегистрированного);
7) дизайнер онлайн магазина;
8) разработчик новой продукции;
9) кассир (выдаёт зарплату сотрудникам);
10) производитель продукции.

ЛАБОРАТОРНАЯ РАБОТА № 4

Протокол уровня защищённых сокетов (SSL protocol)

Цель работы

Цель работы – изучение принципов работы протокола уровня защищённых сокетов (Secure Socket Layers, SSL) на примере СУБД Microsoft SQL Server.

Теоретический материал

Протокол уровня защищённых сокетов (SSL) – криптографический протокол, предназначенный для создания конфиденциального канала связи. Протокол был создан компанией Netscape Communications и применён в конце девяностых годов в рамках веб-браузера Netscape Navigator для создания защищённого HTTP-соединения (HTTPS). На данный момент доказано, что этот протокол является уязвимым для атак, для решения чего был разработан протокол TLS (Transport Layer Security, защита на транспортном уровне), который сейчас находится в практическом использовании. Часто название SSL упоминается в публичных источниках информации как устоявшееся для защищённых соединений, хотя за этим названием стоит реализация TLS. Подавляющее большинство современных браузеров поддерживают протокол TLS как минимум версии 1.0. TLS в целом основан на тех же принципах, что и SSL, предназначен абсолютно для тех же целей, что и SSL, и является просто «работой над ошибками» SSL.

Протокол SSL/TLS основан на двух основных процедурах: протокол «рукопожатия» (handshake) и протокол записи. В ходе первого протокола сервер и клиент устанавливают соединение, проверяют валидность (иногда друг друга, иногда только сервера, изредка сохраняется полная анонимность) и настраивают сеансовый ключ, в ходе протокола записи идёт передача сообщений. Протокол записи достаточно прост: на основе определённого сеансового ключа (который является симметричным) идёт шифрование данных, передача их другой стороне и расшифрование данных при получении.

Протокол «рукопожатия», в общем виде, включает в себя следующие основные шаги:

1) клиент подключается к серверу с запросом на защищённое соединение, предоставляя список поддерживаемых алгоритмов шифрования;

2) сервер выбирает алгоритм из списка (наиболее надёжный и при этом доступный на сервере) и отправляет решение клиенту, вместе с сертификатом для аутентификации себя;

3) клиент проверяет валидность полученного сертификата (путём обращения к указанному в сертификате центру сертификации и сверкой центра сертификации со списком доверенных центров);

4) клиент и сервер создают общий сеансовый ключ (на основе, например, протокола Диффи-Хеллмана).

Соответственно, для создания SSL-соединения необходимы удостоверяющий сертификат сервера на сервере и наличие сертификата удостоверяющего центра сервера в списке доверенных центров сертификации у клиента.

Ход работы

В ходе работы будет выполнена задача соблюдения конфиденциальности данных в процессе передачи данных по открытым каналам связи с задействованием протокола SSL.

Обеспечение защищённого канала связи – непростая задача. Технически защищённый канал связи экономически оправдан только в рамках внутренней сети организации, но за её пределами проводить отдельную защищённую линию связи крайне сложно. Для защиты открытых соединений, проходящих через общую сеть, от перехвата используются криптографические протоколы, превращающие открытый текст в зашифрованный и обеспечивающие уверенность в том, что соединение происходит с доверенным сервером. Одним из таковых протоколов является SSL (TSL).

Рассматриваемая задача разделяется на следующие шаги:

1) получение сертификата, отвечающего требованиям;

2) введение сертификата в окружение сервера и включение режима шифрования;

3) проверка настроенного режима.

Шаг 1

SQL Server реализует протокол SSL с помощью сертификатов, следовательно, асимметричного шифрования. При установке сертификата на сервере необходимо, чтобы сертификат удовлетворял следующим требованиям:

– время сертификата валидно (системное время компьютера находится между значениями времени валидности сертификата);

– сертификат выдан субъекту, название которого полностью совпадает с полным доменным именем сервера или именем компьютера-сервера;

– использование ключа сертификата включает в себя аутентификацию сервера;

– ключ допускает обмен и экспорт (создан с параметром AT_KEYEXCHANGE);

– наличие закрытого ключа в сертификате.

Данные требования НЕ включают в себя требование вида сертификата – возможно использование любого из них. В рамках хода данной лабораторной работы будет создан и использован самоподписанный сертификат, однако следует понимать, что такой сертификат допустимо использовать только тогда, когда нет необходимости в установлении доверенного соединения с сервером – а это только два случая: тестирование сервера и использование протокола в доверенной сети (внутренняя сеть организации, например).

Создать сертификат возможно встроенными средствами Microsoft Windows. Ранее для этого использовалось приложение MakeCert, однако начиная с Windows 10 (Windows Server 2016) функционал по генерации сертификатов входит в расширенную командную строку Windows PowerShell.

Для начала работы запустите Windows PowerShell, обязательно от имени администратора системы. Введите три следующие команды:

```
New-SelfSignedCertificate -DnsName WIN-8PB5C78B03Q -KeySpec
KeyExchange
$Password = ConvertTo-SecureString -String “12345” -Force -AsPlainText
Export-PfxCertificate -Cert cert:\LocalMachine\My\[Отпечаток_сертификата] -
FilePath [Путь_для_сохранения] -Password $Password
```

Первая команда создаёт самоподписанный сертификат без закрытого ключа и со спецификацией ключа на обмен (остальные требования к сертификату SQL Server уже удовлетворены за счёт настроек по умолчанию) с именем виртуальной машины, вторая записывает в переменную Password пароль 12345 в формате защищённой строки (Secure String), третья экспортирует созданный сертификат по указанному пути (не забудьте указать название сертификата и расширение .pfx) с созданием закрытого ключа, защищённого заданным паролем.

Отпечаток сертификата будет выведен в консоль (под словом Thumbprint), однако, чтобы не переписывать его весь, можно открыть созданный сертификат в хранилище сертификатов и скопировать отпечаток из его состава. Чтобы открыть хранилище, с помощью окна запуска Win+R запустите mmc. Откроется окно оснасток, добавьте оснастку «Сертификаты», при определении хранилища выберите «Учётная запись компьютера», далее «Локальный компьютер». Откроется хранилище сертификатов. В хранилище найдите созданный сертификат – он будет в папке «Промежуточные корневые центры сертификации», подпапке «Сертификаты». Откройте его (двойным кликом ЛКМ) и на вкладке «Состав» найдите параметр «Отпечаток» (рис. 3.2), который можно скопировать. Необходимо будет удалить пробелы из отпечатка после вставки, но эта процедура гарантирует, что вы не ошибётесь при переписывании отпечатка из консоли вручную.

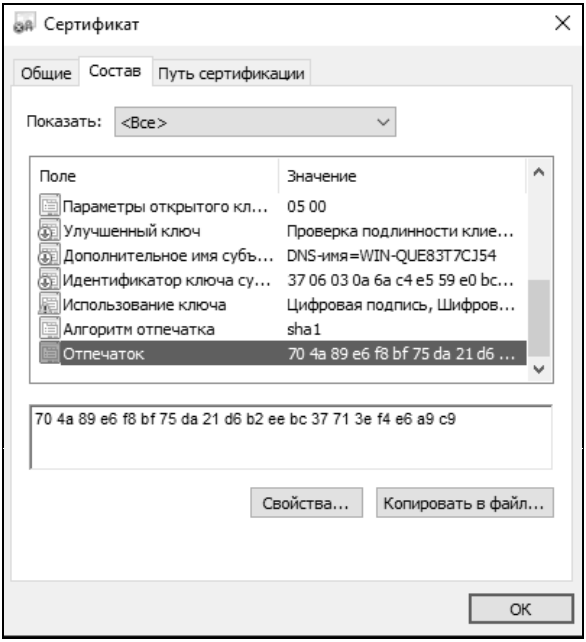


Рис. 3.2. Отпечаток сертификата

Шаг 2

Итак, у нас есть сертификат с закрытым ключом. Теперь необходимо импортировать его в хранилище. Для этого просто его запустите, и откроется Мастер импорта сертификатов (рис. 3.3).

В первом окне Мастера укажите в качестве хранилища локальный компьютер. Второе окно оставьте без изменений. В третьем окне введите пароль, указанный при определении переменной пароля (12345 в примере) (рис. 3.4).

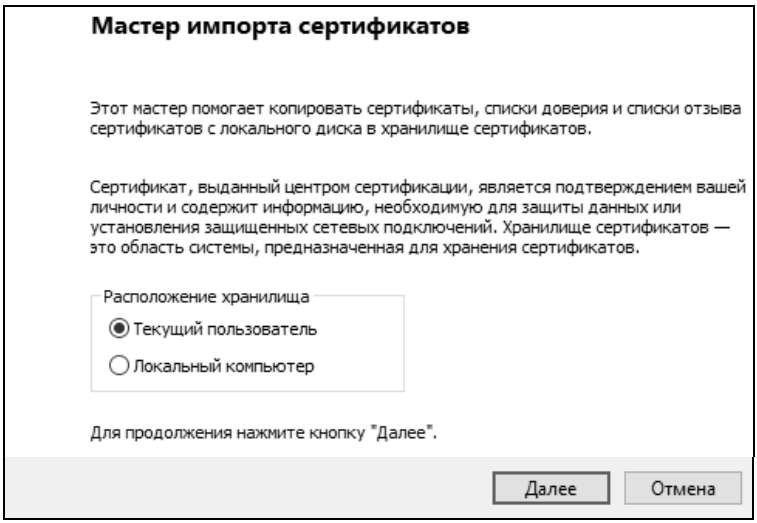


Рис. 3.3. Мастер импорта сертификатов

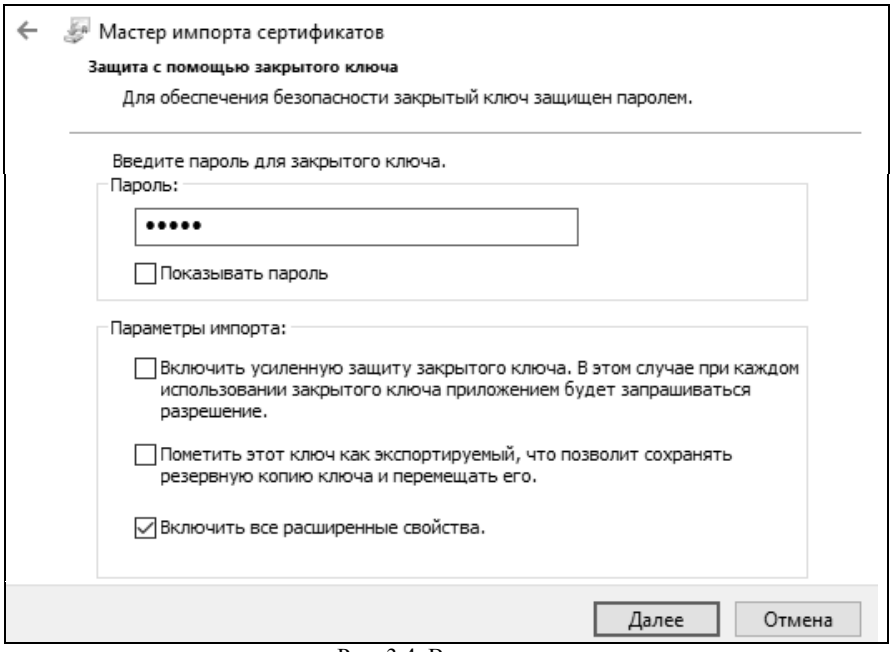


Рис. 3.4. Ввод пароля

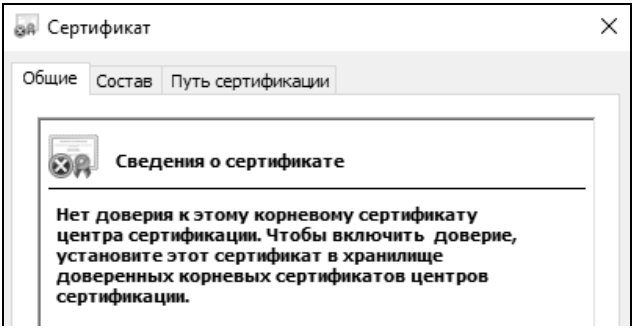
Остальные параметры в третьем окне оставьте без изменений.

В четвёртом окне Мастер предложит указать папку (хранилище), куда будет положен сертификат. По умолчанию сертификат будет положен в папку «Личное» – именно там и должен находиться наш сертификат.

Теперь необходимо выдать доступ службе Microsoft SQL Server к этому сертификату. Для этого, находясь в окне хранилища сертификатов, в контекстном меню этого сертификата (который должен быть в папке «Личное») выберите последовательно пункты «Все задачи» -> «Управление закрытыми ключами». В открывшемся окне укажите пользователя NT Service/MSSQL$SQLSERVER и определите права на полный доступ. Если пункта «Управление закрытыми ключами» нет, значит, ключ находится в хранилище текущего пользователя, и его необходимо поместить в хранилище локального компьютера.

Чтобы сертификат мог быть определён клиентом как допустимый, необходимо положить сертификат удостоверяющего центра (в нашем случае сертификат самоподписанный, следовательно, это будет тот же самый сертификат) в хранилище «Доверенные корневые центры сертификации». Созданный нами изначально сертификат, который находится в папке «Промежуточные центры сертификации», содержащий только открытый ключ, как раз является таким сертификатом. Необходимо перенести этот сертификат в папку доверенных центров в окне хранилища сертификатов.

Теперь в хранилище есть два созданных сертификата – убедитесь, что у сертификата в папке доверенных центров нет закрытого ключа, а в личной папке – есть, что отображается на вкладке «Общие» в окне сертификата (рис. 3.5 и 3.6).



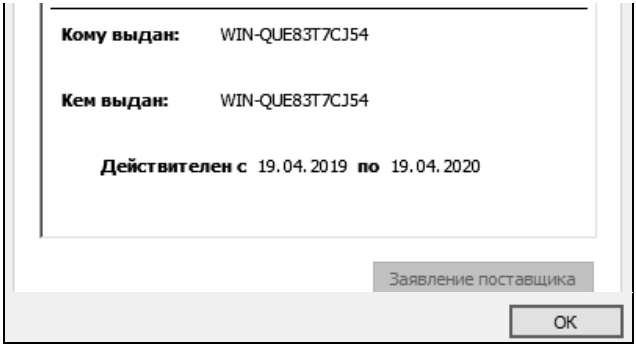


Рис. 3.5. Сертификат без закрытого ключа

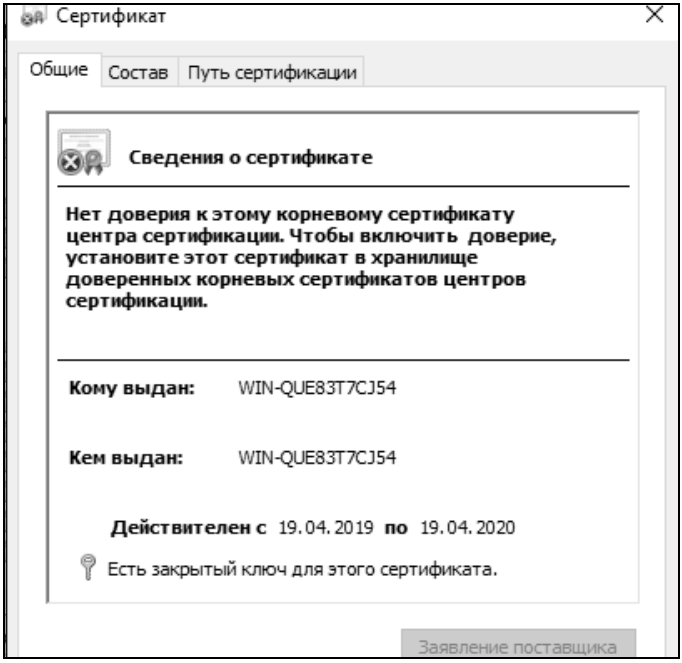


Рис. 3.6. Сертификат с закрытым ключом

Итак, нужные сертификаты добавлены в хранилище. Теперь необходимо указать SQL Server, какой именно сертификат следует использовать для шифрования соединений. Чтобы это сделать, откройте Диспетчер конфигурации SQL Server и с помощью контекстного меню откройте свойства вклад-

ки «Протоколы для MSSQLSERVER» (рис. 3.7). В окне свойств на вкладке «Флаги» включите «Принудительное шифрование», а на вкладке «Сертификат» – укажите добавленный сертификат. Если сертификат не отображается для выбора, где-то была допущена ошибка в создании и/или добавлении.

Шаг 3

Чтобы проверить соединение, необходимо от хоста (основной машины) или другой виртуальной машины, с настроенным соединением его с виртуальной машиной-сервером, присоединиться к серверу с помощью Microsoft SQL Server Management Console (SSMS), или иному приложению, с помощью которого такое соединение возможно. В данной лабораторной работе покажем соединение на примере SSMS.

В первую очередь необходимо добавить сертификат доверенного центра сертификации сертификата сервера (поскольку это самоподписанный сертификат, это опять тот же самый созданный нами сертификат) в хранилище сертификатов на стороне клиента. Для этого следует экспортировать сертификат с сервера на клиент и добавить его в хранилище. Для этого достаточно сертификата с открытым ключом (без закрытого), но можно взять и сертификат с закрытым ключом. Добавляется сертификат в хранилище так же, как и на стороне сервера, однако обратите внимание, что сертификат необходимо добавить в папку доверенных сертификатов (рис. 3.8).

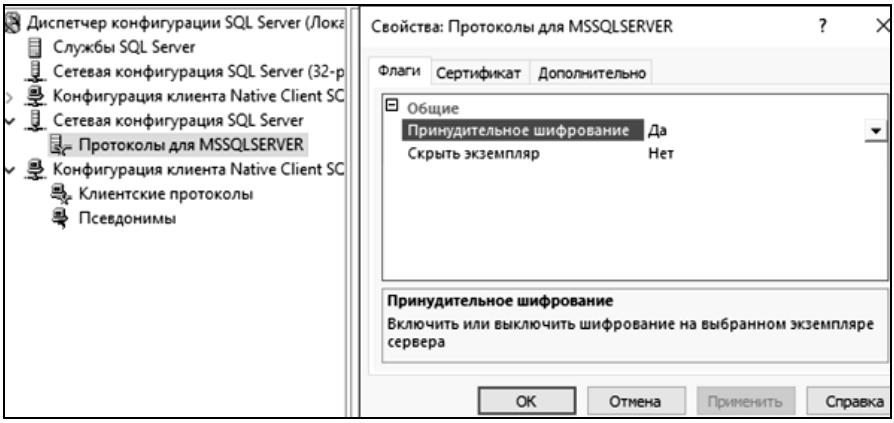
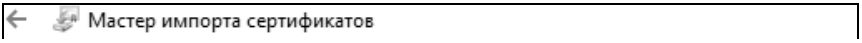


Рис. 3.7. Свойства протоколов



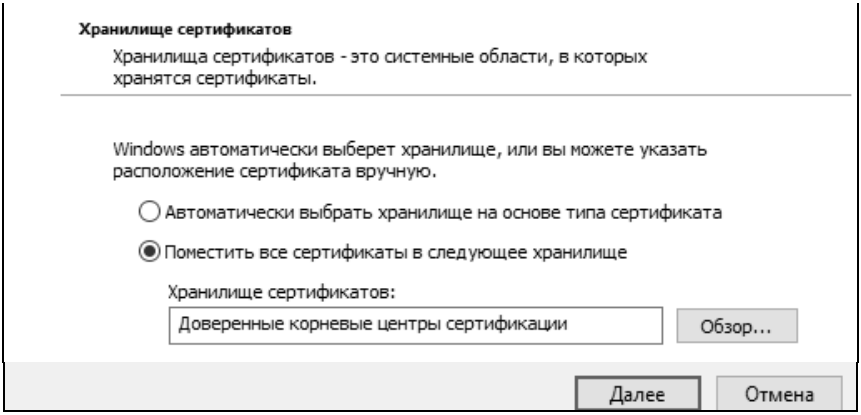


Рис. 3.8. Определение хранилища для сертификата

Теперь необходимо соединиться с сервером. Для этого, при подключении к базе данных, в качестве имени сервера используйте «tcp:WIN-QUE83T7CJ54», что означает подключение к машине с указанным именем по протоколу TCP (рис. 3.9). Поскольку клиент не добавлен в список пользователей СУБД, воспользуйтесь учётной записью sa (пароли указаны в файле «Пароли» на рабочем столе сервера) или иной другой учётной записью с входом по логину/паролю.

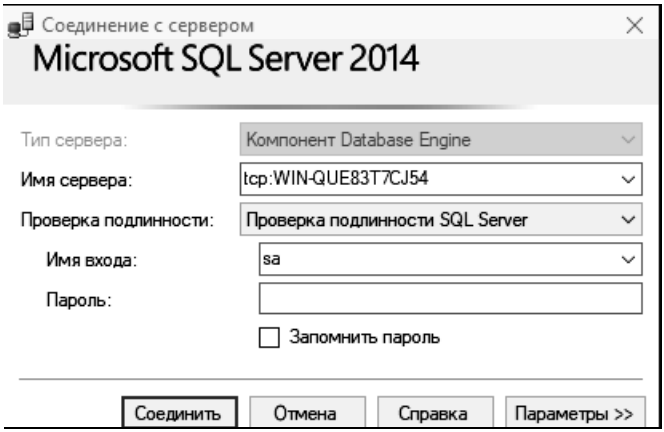
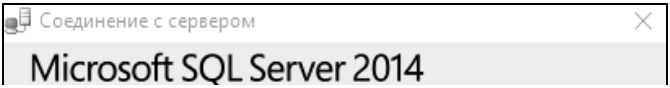


Рис. 3.9. Соединение с сервером



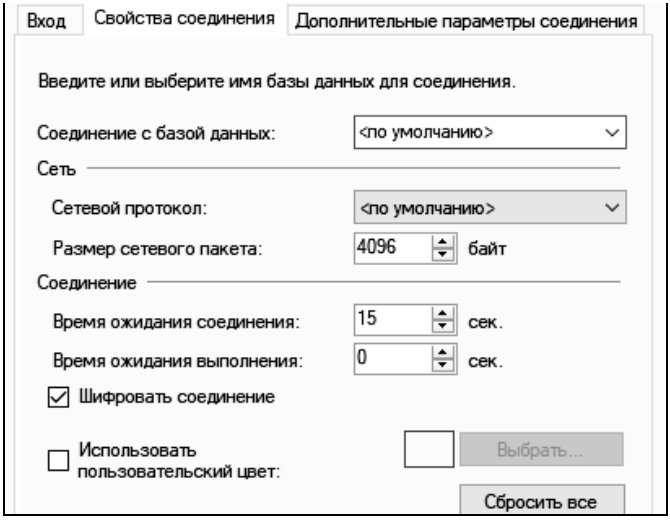


Рис. 3.10. Принудительное применение шифрования со стороны клиента

В зависимости от настроек на сервере, клиент может подключаться как с защищённым соединением, так и без него. Указанный ранее параметр «Принудительное шифрование» уже настроил обязательное шифрование со стороны сервера. Чтобы со стороны клиента использовать именно зашифрованное соединение, следует открыть «Параметры >>» в окне соединения и на вкладке «Свойства соединения» отметить пункт «Шифровать соединение» (рис. 3.10). Убедитесь, что удаётся подключиться к серверу с использованием SSL.

Контрольные вопросы

1) Как расшифровывается SSL и TLS? Что такое SSL и в чём принципиальное отличие от TLS?
2) Какие протоколы (фазы) используются в процессе реализации SSL? Кто участвует в SSL-соединении?
3) Опишите шаги первого протокола (фазы) в SSL.
4) Что необходимо для создания SSL-соединения?
5) Какие существуют виды сертификатов? Чем они отличаются друг от друга? Для чего используются?
6) Какие требования предъявляются к сертификатам в SQL Server?

Задание

Выполните Ход работы на виртуальной машине. Составьте отчёт по проделанной работе и продемонстрируйте отчёт преподавателю.

Дополнительное задание: используя любой анализатор траффика (сниффер), например, Wireshark, продемонстрируйте, что без использования SSL-соединения данные передаются в открытой форме, а с ним они зашифрованы. Учтите, что могут передаваться данные, зашифрованные в рамках первой лабораторной работы, поэтому передавайте незашифрованные данные или расшифруйте ранее зашифрованные.

ЛАБОРАТОРНАЯ РАБОТА № 5

Маскирование

Цель работы

Целью работы является изучение принципов маскирования данных на примере СУБД Microsoft SQL Server.

Теоретический материал

Маскирование данных – процесс преобразования конфиденциальных данных в форму, не позволяющую получить доступ к конфиденциальной части данных, но по-прежнему позволяющую работать с этими данными в иных, кроме ознакомления, целях. Примером таких целей могут быть сортировка данных, идентификация и аутентификация пользователя (например, по четырём первым цифрам номера кредитной карты), сбор статистической информации и пр. Маскирование применяется в тех случаях, когда необходимо раскрыть только часть данных, расположенных в рамках ячейки (например, первые четыре цифры кредитной карты).

Существуют два вида маскирования: статическое и динамическое. В SQL Server статическое маскирование планировалось Microsoft как полноценная возможность, однако разработка была отменена на уровне внедрения в предварительную версию SSMS. Тем не менее, концепция статического маскирования существует и возможна в рамках других СУБД, а также Microsoft не исключают возможность возвращения к этой концепции в будущем. Об статическом маскировании у Microsoft осталась справочная информация, которую приведём ниже.

Статическое маскирование – процесс однонаправленного (невозможно его обратить) преобразования данных по заданному шаблону. Статическое маскирование происходит на уровне столбцов данных. Ко всем ячейкам в выбранном столбце применяется процесс преобразования данных по заданному шаблону. Применяется статическое маскирование в тех случаях, когда базу данных необходимо с какой-либо целью передать стороннему агенту или отделу организации; в зависимости от того, какую цель преследует получатель данных, будет определяться маскирующая функция. В результате статического маскирования создаётся копия таблицы, которая не содержит никакой информации о том, какие данные были в замаскированных столбцах до преобразования. Статическое маскирование планировалось реализовать с помощью мастера. Схема работы статического маскирования показана на рис. 4.1.

Рис. 4.1. Статическое маскирование

Динамическое маскирование данных – процесс ограничения доступа к конфиденциальным данным за счёт сокрытия части этих данных от непривилегированных пользователей с помощью маски. Маска позволяет скрыть часть данных от непривилегированных пользователей, не позволяя им узнать конфиденциальную информацию, но позволяя им каким-либо образом работать с этими данными. От статического маскирования данных динамическое отличается тем, что данные в самой таблице не меняются, меняется их представление для тех пользователей, которые не имеют доступа к демаскированной версии данных. Динамическое маскирование действует на уровне столбцов. Правила маскирования применяются к результатам запроса, таким образом логика действия приложений не меняется. Маскирование не запрещает доступа к данным (включая отсутствие запрета на запись и изменение данных), но закрывает данные от полного просмотра. Реализуется динамическое маскирование (в SQL Server) с помощью Transact-SQL. Для динамического маскирования доступны четыре функции:

– по умолчанию (default()) – полное маскирование данных в соответствии с типом: строка заменяется на xxxx (или меньшее число x, если строка меньше), число на 0, для даты ставится минимальная дата (1 января 1900 года), для двоичных данных ставится двоичный 0;

– Email (email()) – маскирование для адреса электронной почты (остаются открытыми первая буква электронного адреса и адреса 0-1 уровня доменного имени, остальные символы заменяются на X);

– случайное (random([start range], [end range]) – маскирование заменой на случайное число из приведённого диапазона [start range : end range] (работает, соответственно, только с числовыми значениями);

– пользовательское (partial(prefix, padding, suffix)) – маскирование, которое открывает заданное число символов по краям (prefix, suffix) строки данных и закрывает символы посередине заданной строкой (padding).

Невозможно динамическое маскирование для следующих столбцов:

– зашифрованные с помощью постоянного шифрования;

– FILESTREAM;

– часть набора столбцов COLUMN_SET;

– вычисляемый столбец (но немаскированный вычисляемый столбец может быть основан на маскированных столбцах, в таком случае он будет возвращать результат на основании маскированного значения);

– имеет зависимости (чтобы это обойти, можно удалить зависимости, создать маску и вернуть зависимости).

– также столбец с динамическим маскированием не может применяться в качестве ключа для полнотекстового индекса.

Важно обратить внимание на то, что динамическое маскирование не защищает от запросов с условиями. Выведенные данные будут замаскированными, но их можно вывести статистически, например, запросом:

```
SELECT ID, Name, Salary FROM Employees
WHERE Salary > 99999 and Salary < 100001;
```

можно получить всех пользователей с зарплатой 10000, хотя в результативной таблице будет показана маскированная зарплата. Необходимо применять маскирование в совокупности с другими средствами разграничения доступа и грамотно задавать права пользователям базы данных.

Ход работы

В ходе работы будет выполнена задача соблюдения конфиденциальности данных в процессе обработки этих данных пользователями базы данных.

Некоторым пользователям базы данных не полного уровня доступа не требуется знание всей информации, предоставленной в таблице. На уровне столбцов и строк разделение прав доступа просто и понятно, но когда необходимо защитить часть столбца данных, оставляя пользователям возможность работать с другой их частью, не имеющей в себе закрытой информации, применяется именно маскирование. Использование маскирования позволяет, например, сотрудникам в службе поддержки аутентифицировать пользователя, спросив у него первые четыре номера кредитной карты – при этом сами сотрудники полного номера знать не обязаны; или позволяет аналитическому отделу собирать статистику по наиболее используемым хостингам почтовых служб – не распространяя информацию о том, у какого пользователя какая почта. Функционал маскирования возможно реализовать с по-

мощью внешних средств – и в рамках однократной выборки данных это ещё можно назвать целесообразным, однако когда маскировать данные необходимо на регулярной основе (как уже в приведённом примере с сотрудником службы поддержки), осуществление этого маскирования на уровень выше (с уровня прикладного приложения на уровне СУБД) уменьшает поверхность атаки, тем самым уменьшая вероятность выполнения атаки (злоумышленник может перехватить данные ещё до обработки их приложением, однако если данные уже замаскированы на уровне СУБД, это злоумышленнику ничего не даст). Если к этому добавить упрощение разработки прикладных приложений, которым более не нужно выполнять это маскирование, преимущества маскирования на уровне СУБД становятся ещё более очевидны. Недостатком такого подхода можно назвать только ограниченность маскирующих функций – что целиком и полностью зависит от разработчика СУБД.

Рассматриваемая задача разделяется на следующие шаги:

1) определение цели маскирования;
2) определение данных, подлежащих маскированию;
3) определение функции маскирования согласно определённой цели и виду данных;
4) осуществление процесса маскирования.

В рамках SQL Server покажем процесс маскирования на примере динамического маскирования.

Шаг 1

В совокупности с другими данными, необходимо обеспечить конфиденциальность номеров кредитных карт покупателей в процессе работы с данными теми сотрудниками, которым нет необходимости знать весь номер кредитной карты, но есть необходимость знать первые четыре цифры этой кредитной карты для подтверждения покупки.

Шаг 2

Необходимо маскировать данные о номерах кредитных карт покупателей. Эти данные расположены в таблице Sales.CreditCard, столбце CardNumber. Запустите SSMS, зайдите в СУБД от имени текущей учётной записи Windows, откройте эту таблицу и ознакомьтесь с её структурой (рис. 4.2).

Шаг 3

Необходимо оставить открытыми первые четыре символа номера и закрыть остальные (десять). Для этого необходимо воспользоваться функцией маскирования partial с параметрами 4, «XXXXXXXXXX» 0.

Шаг 4

Динамическое маскирование реализуется с помощью Transact-SQL. Выполните следующий запрос (в рамках БД AdventureWorks2017):

```
ALTER TABLE Sales.CreditCard
ALTER COLUMN CardNumber ADD MASKED WITH (FUNCTION = 'partial(4, "XXXXXXXXXX", 0)');
```

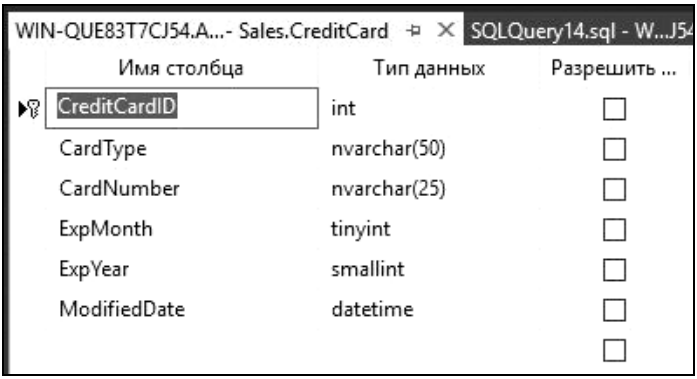


Рис. 4.2. Структура таблицы Sales.CreditCard

Если запрос завершается ошибкой, не исключено, что не отменено шифрование, выполненное в рамках предыдущей лабораторной работы. Отмените его перед выполнением маскирования.

Просмотрите значения в таблице Sales.CreditCard (с помощью пункта «Выбрать первые 1000 строк» в контекстном меню таблицы или с помощью запроса SELECT). Убедитесь в том, что номер карты отображается незамаскированным (рис. 4.3).

Результаты | Сообщения

	CreditCardID	CardType	CardNumber	ExpMonth	ExpYear	ModifiedDate
1	1	SuperiorCard	33332664695310	11	2006	2013-07-29 00:00:00.000
2	2	Distinguish	55552127249722	8	2005	2013-12-05 00:00:00.000
3	3	ColonialVoice	77778344838353	7	2005	2014-01-14 00:00:00.000
4	4	ColonialVoice	77774915718248	7	2006	2013-05-20 00:00:00.000
5	5	Vista	11114404600042	4	2005	2013-02-01 00:00:00.000
6	6	Distinguish	55557132036181	9	2006	2014-04-10 00:00:00.000
7	7	Distinguish	55553635401028	6	2007	2013-02-01 00:00:00.000
8	8	SuperiorCard	33336081193101	7	2007	2013-06-30 00:00:00.000
9	9	Distinguish	55553465625901	2	2005	2013-09-23 00:00:00.000
10	10	SuperiorCard	33332126386493	8	2008	2011-08-31 00:00:00.000
11	11	SuperiorCard	33335352517363	10	2008	2014-05-04 00:00:00.000
12	12	SuperiorCard	33334316194519	4	2008	2012-05-30 00:00:00.000
13	13	Vista	11119775847802	11	2005	2014-03-01 00:00:00.000
14	14	Distinguish	55553287727410	10	2006	2013-11-19 00:00:00.000
15	15	SuperiorCard	33336866065599	11	2008	2013-01-29 00:00:00.000
16	16	Vista	11111985451507	8	2006	2013-12-30 00:00:00.000

Рис. 4.3. Незамаскированные данные

Это происходит потому, что у вашей текущей учётной записи (sa или вход с помощью аутентификации Windows) есть полные права на просмотр данных, включая право на просмотр немаскированных данных. Чтобы проверить, замаскированы ли данные, просмотрите их от имени другого пользователя. Для этого можете создать временного пользователя без имени входа, выдать ему право SELECT на рассматриваемую таблицу и выполнить этот запрос от его имени. Или же создать полноценную пару имя входа-пользователь, выдать пользователю право SELECT, зайти по созданным именем входа и выполнить этот запрос. Выполнив это, убедитесь, что данные действительно замаскированы (рис. 4.4).

100 %

Результаты | Сообщения

	CreditCardID	CardType	CardNumber	ExpMonth	ExpYear	ModifiedDate
1	1	SuperiorCard	3333XXXXXXXXXX	11	2006	2013-07-29 00:00:00.000
2	2	Distinguish	5555XXXXXXXXXX	8	2005	2013-12-05 00:00:00.000
3	3	ColonialVoice	7777XXXXXXXXXX	7	2005	2014-01-14 00:00:00.000
4	4	ColonialVoice	7777XXXXXXXXXX	7	2006	2013-05-20 00:00:00.000
5	5	Vista	1111XXXXXXXXXX	4	2005	2013-02-01 00:00:00.000
6	6	Distinguish	5555XXXXXXXXXX	9	2006	2014-04-10 00:00:00.000
7	7	Distinguish	5555XXXXXXXXXX	6	2007	2013-02-01 00:00:00.000
8	8	SuperiorCard	3333XXXXXXXXXX	7	2007	2013-06-30 00:00:00.000
9	9	Distinguish	5555XXXXXXXXXX	2	2005	2013-09-23 00:00:00.000
10	10	SuperiorCard	3333XXXXXXXXXX	8	2008	2011-08-31 00:00:00.000
11	11	SuperiorCard	3333XXXXXXXXXX	10	2008	2014-05-04 00:00:00.000
12	12	SuperiorCard	3333XXXXXXXXXX	4	2008	2012-05-30 00:00:00.000
13	13	Vista	1111XXXXXXXXXX	11	2005	2014-03-01 00:00:00.000
14	14	Distinguish	5555XXXXXXXXXX	10	2006	2013-11-19 00:00:00.000
15	15	SuperiorCard	3333XXXXXXXXXX	11	2008	2013-01-29 00:00:00.000
16	16	Vista	1111XXXXXXXXXX	8	2006	2013-12-30 00:00:00.000

Рис. 4.4. Замаскированные данные

Не все пользователи должны видеть замаскированное значение. Право на просмотр немаскированных значений выдаётся пользователю или роли и распространяется на всю базу данных целиком, без спецификации конкретных столбцов и таблиц. Чтобы дать пользователю разрешение на просмотр незамаскированных данных, выдайте ему право UNMASK:

```
GRANT UNMASK to [имя_пользователя];
```

Аналогичным образом, с помощью команды REVOKE, разрешение отменяется.

Выдайте созданному ранее пользователю право на просмотр немаскированных значений. Убедитесь, просмотрев таблицу с маской от его имени, что значения действительно отображаются незамаскированными.

Изменение бизнес-процессов в компании, или, возможно, необходимость добавить новые связи в таблицу, могут привести к необходимости изменить или удалить маскирование. Для изменения функции маскирования необходимо воспользоваться командой изменения столбца с ключевым словом MASKED и указанием функции маскирования (практически аналогично добавлению маски, только без слова ADD):

```
ALTER COLUMN [имя_столбца] MASKED WITH (FUNCTION = '[функция]');
```

Для того же, чтобы удалить маску, необходимо выполнить следующую команду:

```
ALTER COLUMN [имя_столбца] DROP MASKED
```

Проверьте на выбранном столбце ещё какую-нибудь маскирующую функцию (например, random([0, 999999])). Удалите маску со столбца.

Контрольные вопросы

1) Что такое маскирование?
2) В чём различия между статическим и динамическим маскированием?
3) Для чего чаще всего используется статическое маскирование? Для чего динамическое?
4) Какие функции динамического маскирования есть в SQL Server?
5) Что позволяет динамическое маскирование (в рамках прав доступа) в SQL Server? В чём его уязвимость относительно этого?
6) Каким образом создаётся, изменяется и удаляется динамическая маска в SQL Server?

Задание

Проанализируйте ситуацию, данную в вашем варианте, и определите столбцы (столбец), требующие маскирования, и режим маскирования. Примените выбранное маскирование к столбцам. Укажите, кому следует иметь доступ к незамаскированным значениям. Объясните выбор столбцов и субъектов, имеющих доступ к незамаскированным значениям. Напишите отчёт по проделанной работе и защитите отчёт у преподавателя.

Варианты задания:

1) При написании обзоров на товары компании Adventure Works покупатели оставляли своё имя и электронную почту. Целью сбора этих электронных адресов является оповещение о новых товарах и акциях. Чтобы использовать их в других целях, например, статистических, необходимо эти адреса обезличить.
2) Данные о сотрудниках нужны практически всему начальственному составу организации и практически всему отделу кадров. Если многие данные о сотрудниках действительно нужны для работы, то номер паспорта в бизнес-процессе нужен далеко не всем. Правда, в рамках компании, поскольку она расположена в Америке, используется несколько другой номер – social security number, который является частным случаем более общего понятия – national identification number.
3) С помощью, в том числе, номера отслеживания состояния доставки товара многие перевозчики осуществляют аутентификацию заказчика. Знание этого номера может позволить злоумышленнику перехватить какой-либо заказ, заявив себя как представитель покупателя.

4) Заработные платы сотрудников являются их личной тайной. Время от времени все экономические данные в базе передаются отделу аналитики. Точные цифры заработных плат им в том числе нужны. Однако знать, какому человеку какие заработные платы соответствуют, им нет нужды.

5) На основании электронных адресов в организации осуществляется подтверждение аутентификации в системе – как для пользователей, так и для сотрудников. При подтверждении, однако, нет никакой нужды подтверждающему знать полный адрес.

Дополнительное задание: проанализируйте структуру СУБД AdventureWorks2017 на предмет возможных применений и определите ещё как минимум 5 таблиц, к которым возможно осмысленное применение маскирования. Укажите режим маскирования, столбцы, к которым следует применять маскирование, и категорию пользователей, которые должны иметь доступ к незамаскированным значениям. Сделанный выбор обоснуйте.

ЛАБОРАТОРНАЯ РАБОТА №6

Оценка уязвимостей и классификация данных

Цель работы

Целью работы является изучение принципов работы автоматизированных оценщиков уязвимостей и классификаторов данных на примере соответствующих инструментов в SQL Server.

Теоретический материал

Инструмент оценки уязвимостей (Vulnerability Assessment) в SQL Server основывается на наборе предопределённых правил. Правило представляет собой рекомендацию от корпорации Microsoft по соблюдению информационной безопасности. Все правила проверяются при выполнении проверки, в результате которой определяется, какие правила соблюдены. Структурно правило состоит из следующих компонентов:

- номер (ID);
- название;
- уровень угрозы (риска) (высокий, средний, низкий);
- статус (проверка пройдена или провалена);
- описание правила;
- влияние угрозы в правиле;
- запрос – обычный запрос на чтение (SELECT) к базе данных, по результатам которого и определяется, соблюдено ли правило или нет;
- рекомендация от Microsoft к правилу;
- фактический результат запроса;
- решение;
- скрипт решения (если доступен).

У правил существует понятие базового показателя – значения, которое считается нормой, а нарушением правила будет отклонение от такого значения. Примером правила, которое основано на базовом показателе, является правило с ID VA1281, которое заключается в проверке того, что все членства пользователей в пользовательских ролях учтены в этом базовом показателе – такое правило позволяет динамически отслеживать изменение состава ролей в базе данных.

Всего таких проверок 56 на обычные базы данных. Также присутствует оценка уязвимостей на системные базы данных, на каждую из которых

прописан уникальный набор проверок (кроме tempdb, на которую проверок нет), хотя многие правила повторяются.

Средства классификации данных предназначены для обозначения уязвимой, критичной и прочей важной информации в рамках баз данных. Такие средства чаще всего используются в совокупности с другими средствами, предлагая применить те или иные меры на классифицированные так или иначе столбцы. Правда, SQL Server таких предложений не делает. Как и средства поиска уязвимостей, классификация данных предназначена больше для начинающего администратора безопасности, но также полезна в рамках больших баз данных, когда необходимо быстро определить потенциально уязвимые столбцы, а времени на ознакомление со всем бизнес-процессом и, соответственно, ручного определения таких данных не хватает. Также это средство, как и оценка уязвимостей, очень полезно, когда окружение (структура баз данных) меняется динамически и довольно быстро. В целом, они помогают быстро построить основу, по которой можно также быстро принять решение о принятии базовых мер по информационной безопасности данных, и в конце, опять же быстро, сформировать отчёт по сделанным мерам и по ситуации.

Одно из правил оценки уязвимостей, а именно правило VA1285, предлагает идентифицировать все столбцы с конфиденциальной информацией. Это напрямую связывает оценку уязвимостей с классификацией данных. Обнаружение и классификация данных (Data discovery and classification) – инструмент, предназначенный для пометки уязвимых столбцов для более удобной работы с такими столбцами. SQL Server позволяет автоматически определить столбцы с уязвимыми данными и затем в ручном режиме подтвердить/отказать и дополнить построенную классификацию. Столбцы помечаются различными метками (персональные данные, кредитная карта, финансы, аутентификационная информация и пр.). После определения уязвимых столбцов и их меток инструмент сформирует отчёт о классификации данных, с использованием круговых диаграмм и с перечислением уязвимых данных.

Ход работы

В ходе работы будет выполнена задача обеспечения безопасности данных путём применения автоматизированных средств оценки уязвимостей и классификации данных.

В различных окружениях существуют различные средства автоматизации. В рамках СУБД и обеспечения информационной безопасности также существуют средства, позволяющие так или иначе облегчить процесс защиты данных. Такие средства помогают познакомить пользователя с имеющимися в СУБД средствами защиты, обнаружить типичные уязвимости, присутствующие в установке по умолчанию, указать на неверные способы выдачи прав

доступа, определить содержащие потенциально конфиденциальную информацию столбцы, и прочее. Такие средства больше предназначены для начинающих администраторов безопасности, поскольку просты в применении. Однако никакое средство автоматизации поиска уязвимостей не обнаружит все реально существующие уязвимости, находящиеся в их окружении; каждое из них ограничено тем кругом поиска, на который их настроили создатели. И уж совершенно точно такие средства не могут самостоятельно, без обращения к внешним источникам, обнаруживать программные уязвимости. Однако в рамках большого количества крупных баз данных такие средства позволят осуществлять быструю проверку на применение самых основных мер безопасности в каждой из них, а также помогают обнаружить те уязвимости, которые можно обнаружить автоматизированным путём, но сложно обнаружить человеку, например, некоторые преувеличения разделения прав доступа.

Рассматриваемая задача разделяется на следующие шаги:

1) запуск автоматизированного средства оценки уязвимостей;
2) оценка полученного списка проблем и определение мер по устранению;
3) принятие определённых мер;
4) составление списка информации критичной важности.

Шаг 1

Оценка уязвимостей в SQL Server происходит на уровне базы данных. Чтобы запустить оценку уязвимостей, следует в контекстном меню базы данных выбрать "Задачи" -> "Оценка уязвимостей" -> "Проверить на наличие уязвимостей...". Откроется окно выбора места сохранения результатов оценки, можно оставить значение по умолчанию. После выбора этого места будет произведена оценка, и затем откроется окно результатов оценки (Рисунок 6.1).

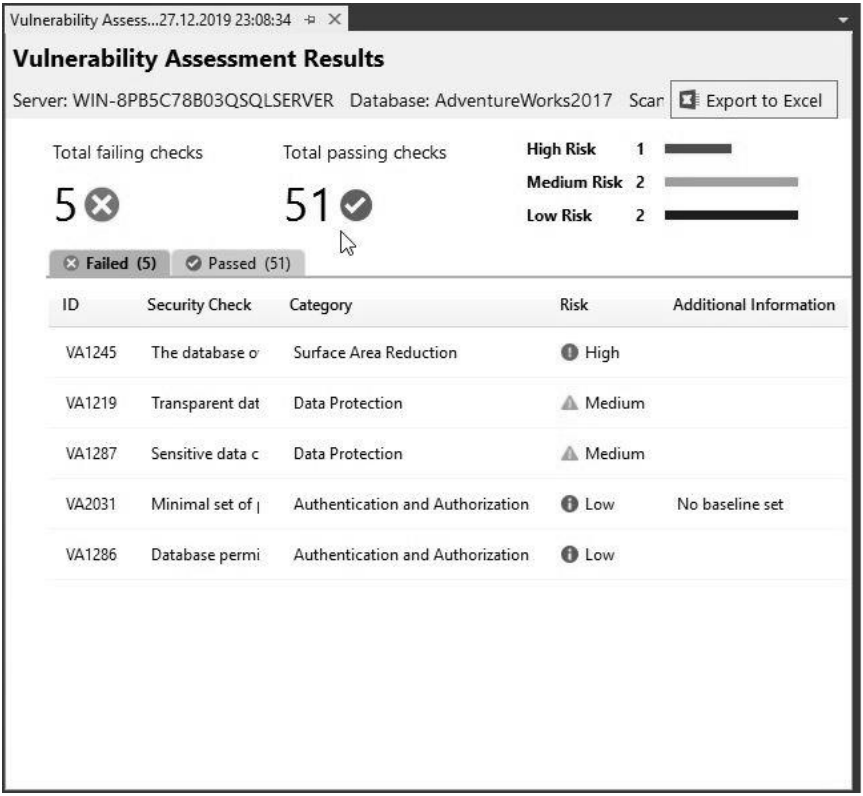


Рисунок 6.1 – Результат оценки

Шаг 2.

Некоторые из правил будут провалены (failed), многие же останутся выполненными (passed). Точный список зависит от того, что и как было выполнено в предыдущих лабораторных работах. Обратимся к одному из правил, а именно правилу с ID VA1286 (Рисунок 6.2).

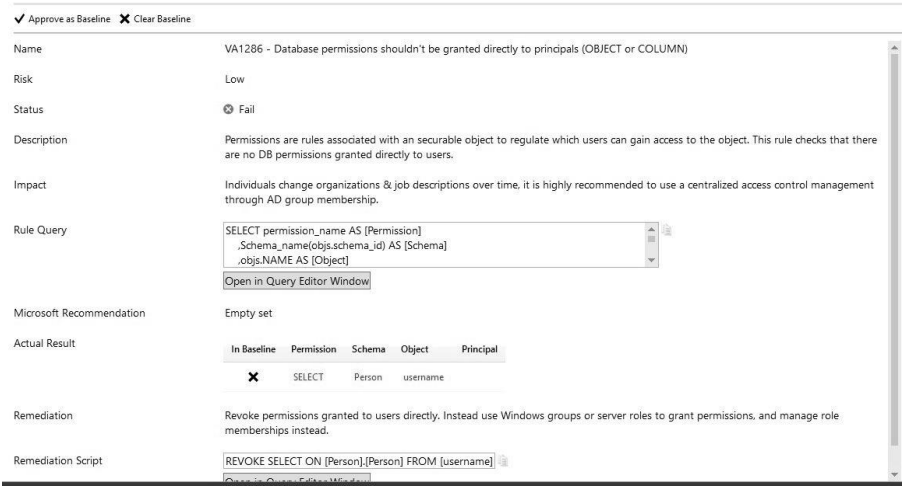


Рисунок 6.2 – Правило

Прочитав это правило, становится понятно, о чём оно, что проверяет, какую проблему затрагивает и какое решение предлагает. В случае этого конкретного правила проверяется, чтобы разрешения на объекты и столбцы не выдавались непосредственно пользователям. Оно не несёт большого риска. Это считается небезопасным не само по себе, а потому что в часто меняющихся реалиях организациях обязанности конкретных людей меняются постоянно, соответственно этому меняются и их права доступа в СУБД. Чтобы более централизованно распределять права, рекомендуется выдавать доступ ролям, а не конкретным людям, за чем, в числе других, следит это правило. Рекомендуется отозвать эти права, для чего уже подготовлен скрипт в пункте Remedation Script. Скрипт можно открыть в окне запроса нажатием соответствующей кнопки снизу. Также можно принять текущее состояние как основу, нажав на кнопку Approve as Baseline сверху. Потребуется подтвердить своё решение, прежде чем базовый показатель будет принят. Чтобы увидеть изменения, следует выполнить оценку уязвимостей по новой. Впоследствии у этого правила будет приписка в столбце Additional information о том, что к нему задан пользовательский базовый показатель.

Шаг 3.

Рассмотрев таким образом все правила, примите меры к решению проблем.

Шаг 4.

Одно из правил наверняка провалено, а именно правило с ID VA1287. Это правило строится вокруг классификации данных, и его суть проста – критически важные данные должны быть классифицированы. В рамках оценки уязвимостей будет предложен набор автоматически выбранных (на основании названия) столбцов. Чтобы открыть средство классификации данных в отдельном окне, следует выбрать его в контекстном меню нужной базы данных ("Задачи" -> "Обнаружение и классификация данных" -> "Классификация данных..."). Откроется окно классификации (Рисунок 6.3).

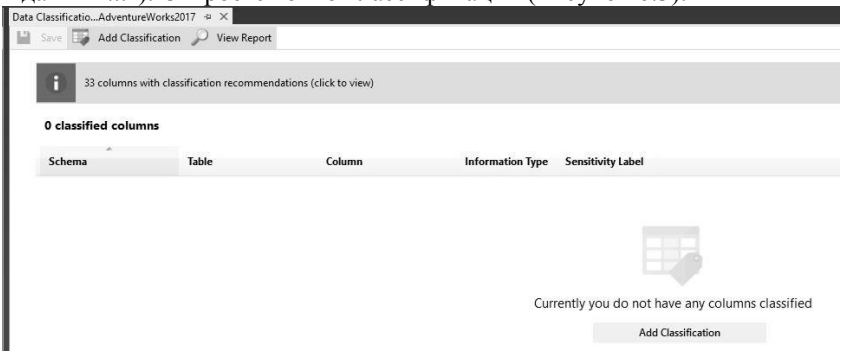


Рисунок 6.3 – Классификация данных

Изначально классифицированных данных нет, однако СУБД предлагает автоматически найденные столбцы, которых в рамках БД AdventureWorks2017 оказывается 33. Просмотрите предложенную классификацию, кликнув по окну с сообщением (Рисунок 6.4).

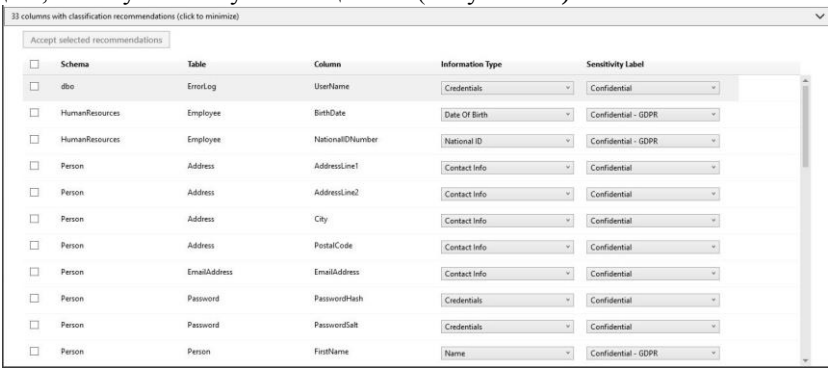


Рисунок 6.4 – Предложенная классификация

Каждая классификация состоит из следующих частей: схема, таблица, столбец, тип информации, метка уязвимости. Тип информации относится к тому, чем является информация (дата рождения, персональные данные, контактные данные, финансовые данные и т.д.), меткой уязвимости определяется, насколько информация критична (публичная, общая, конфиденциальная, персональные данные (GDPR – General Data Protection Regulation, постановление Европейского союза, регулирующее защиту персональных данных), высоко конфиденциальная, высоко конфиденциальные ПД и без категории). На основании сгенерированной классификации можно поменять тип информации и метку уязвимости, удалить лишние классификации, а также добавить новые с помощью меню справа (Рисунок 6.5).

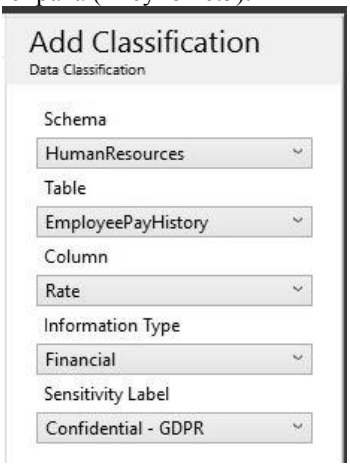
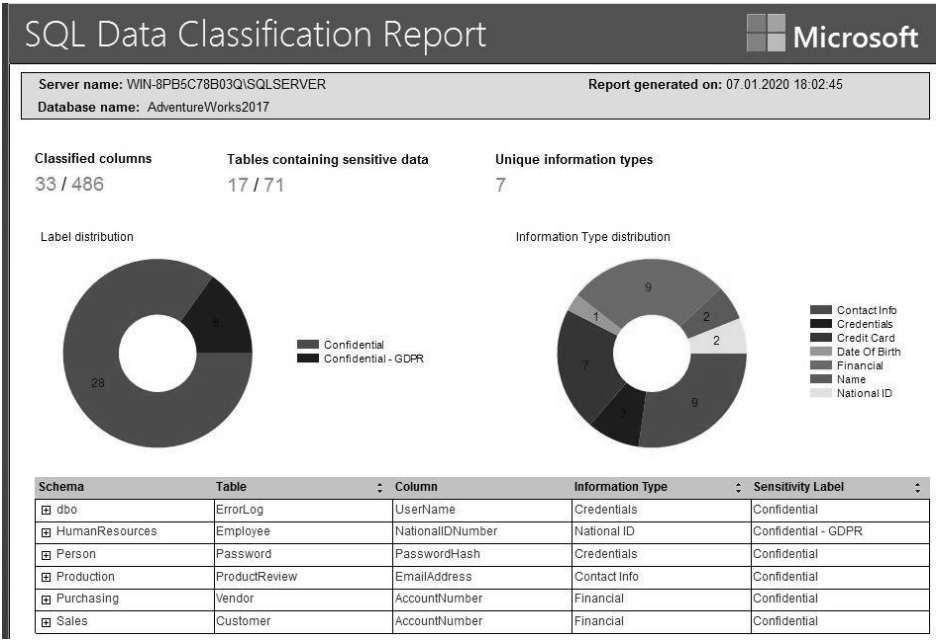


Рисунок 6.5 – Добавление классификации

После завершения работы над классификацией необходимо сохранить изменения. Далее производится генерация отчёта, в котором будет отмечено на диаграммах, сколько было сделано классификаций и по каким категориям они распределены, а также представлена таблица с классификациями (Рисунок 6.6).



Schema	Table	Column	Information Type	Sensitivity Label
⊞ dbo	ErrorLog	UserName	Credentials	Confidential
⊞ HumanResources	Employee	NationalIDNumber	National ID	Confidential - GDPR
⊞ Person	Password	PasswordHash	Credentials	Confidential
⊞ Production	ProductReview	EmailAddress	Contact Info	Confidential
⊞ Purchasing	Vendor	AccountNumber	Financial	Confidential
⊞ Sales	Customer	AccountNumber	Financial	Confidential

Рисунок 6.6 – Отчёт по классификации данных

Контрольные вопросы

1) Дайте определение автоматизированной оценке уязвимостей;
2) Каковы основные недостатки автоматизированной оценки уязвимостей? Каковы её достоинства?
3) Когда наиболее эффективно использовать автоматизированную оценку уязвимостей?
4) На чём основываются правила в оценке уязвимостей SQL Server?
5) Что такое базовый показатель (baseline) в оценке уязвимостей SQL Server?
6) Что такое и для чего нужна классификация данных?

Задание

Выполните Ход работы. Также проведите оценку уязвимостей и классификацию данных в базе данных и заданных рамках согласно варианту. Учитывайте, что не все меры и не вся классификация, предоставленные автоматизированными средствами, могут соответствовать описанной в варианте ситуации. Составьте отчёт по проделанной работе, оформите пункт отчёта

с ходом работы в таком виде, в каком бы вы предоставили его своему вышестоящему начальству о проделанной начальной работе по обеспечению безопасности в базе данных. Необходимо указать, какие меры были приняты, в чём они заключались, от чего (каких уязвимостей) они защитили и обозначить критичные данные в базе данных с их классификацией, а также сделать краткий вывод. Также следует проанализировать выполненные правила и составить список проблем, которые уже были учтены. В отчёте используйте автоматически генерируемые отчёты инструментов оценки уязвимостей и классификации данных. Не выходите за рамки средств оценки уязвимостей и классификации данных.

Дополнительное задание: проделайте то же задание, только с системной базой данных master и без классификации данных.

ЛАБОРАТОРНАЯ РАБОТА №7
Защита от SQL-инъекций

Цель работы

Целью работы является изучение принципов реализации атак вида "SQL-инъекция" и методов защиты от атак такого вида.

Теоретический материал

SQL-инъекция (атака посредством внедрения SQL кода) – атака, которая заключается во внедрении вредоносного кода через пользовательский интерфейс, который в процессе обработки внутри приложения будет преобразован в SQL запрос. Уязвимости вида "SQL-инъекция" не относятся к уязвимостям самих СУБД и происходят на уровне приложения. Поскольку СУБД обрабатывает любой синтаксически верный запрос, защита на уровне СУБД от внедрения SQL кода невозможна (возможна с проверкой на "подозрительный" код, однако наверняка отличить обычный код от внедрённого в рамках СУБД невозможно). Однако, поскольку в результате SQL-инъекций, происходит непосредственная утечка информации из СУБД, необходимо рассмотреть её в рамках этого курса лабораторных работ.

В ходе реализации SQL-инъекции злоумышленник вводит в окно пользовательского запроса (это может быть окно приложения, окно ввода данных в веб-сайте, http-запрос к веб-сайту, и т.п.) замаскированный от приложения SQL код, который впоследствии интерпретируется приложением не как переменная в запросе, как предполагалось его разработчиками, а как код, потому что происходит конкатенация этих пользовательских данных с заранее определённой строкой кода SQL запроса. Таким образом злоумышленник может внедрять самые разнообразные запросы в код, получая необходимую ему информацию.

Для лучшего понимания принципа работы SQL-инъекций приведём пример. Допустим, проверка пароля пользователя происходит посредством поиска в таблице Password совпадения введённых пользователем имени и пароля с содержимым столбцов username и password соответственно; если соответствующая запись в таблице есть, то человек проходит аутентификацию (примечание: на практике подобная проверка должна осуществляться принципиально другими, более безопасными, методами, для чего используются хэши и соль паролей). Для этого составляется запрос вида:

SELECT * FROM Password WHERE username = '[user]' and password = '[pass]';

Приложение вместо [user] подставляет введённое пользователем имя, а вместо [pass] – введённый пользователем пароль.

Злоумышленник, каким-либо образом, например, методом проб и ошибок, определил структуру этого запроса и решил вместо указания пароля ввести, например:

1' or username = 'admin

В результате чего получится следующий запрос:

SELECT * FROM Password WHERE username = '' and password = '1' or username = 'admin';

В результате него будет возвращена запись с именем пользователя admin. Если в системе такой пользователь имеется и является администратором, злоумышленник получит доступ к функционалу администратора в приложении. Если же злоумышленник хочет получить полный список пользователей, ему достаточно указать всегда истинное выражение, например:

1' or 1=1; -- ';

Сочетание символов -- (два дефиса) означает комментарий, соответственно, символы '; , которые добавляются приложением в конце запроса и завершают его, будут проигнорированы, вместо них останется только ; , указанный злоумышленником для составления синтаксически верного запроса.

Существует множество различных видов SQL-инъекций и множество различных техник их применения. Применяемые техники не стоят на месте и развиваются вместе с развитием средств защиты. Основные же виды принято делить по получаемой злоумышленником информации, кои и перечислим:

– внедрение операторов SQL – основной вид инъекции, простой пример которой описан выше: злоумышленник внедряет в код свои операторы и получает полноценный ответ на свой запрос;

– слепое внедрение операторов SQL (слепая SQL-инъекция) – вид инъекции, в которой злоумышленник получает в качестве ответа от приложения не результат запроса, а бинарный ответ (да или нет). Соответственно этому внедряемый им код определяется так, чтобы ответ был получен не на тот вопрос, который подразумевается разработчиками приложения, а на вопрос, который ставит злоумышленник. Этот вид инъекции разделяется в зависимости от того, возвращается ли приложением сообщение об ошибке или нет (из сообщения об ошибке можно получить множество ценной для злоумышленника информации);

– двойное слепое внедрение операторов – злоумышленник получает в качестве ответа только время выполнения запроса; результат запроса же всегда одинаков (например, что запрос был обработан).

Основные техники защиты от инъекций включают в себя три вида правил:

1) обнаружение инъекции в рамках приложения при построении кода – набор техник, используемых при программировании приложения, которые будут подробнее рассмотрены далее;

2) минимизация прав доступа пользователя – даже если злоумышленнику удастся внедрить свой код, СУБД не позволит ему исполнить больше, чем пользователь, от имени которого он выступает. Зачастую приложение выступает от имени одного фактического пользователя СУБД, который может всё или почти всё, а разделение прав доступа происходит на уровне приложения. Такой подход развязывает злоумышленникам руки при использовании SQL-инъекций. Техники внедрения инъекций включают в себя в том числе обход техник защиты от них, однако при всём желании одними только SQL инъекциями преодолеть доступные пользователю права злоумышленник не сможет;

3) использование автоматизированных средств обнаружения инъекций (WAF – web application firewall) – средства, обнаруживающие "подозрительный" код и призывающие внимание администратора в случае обнаружения ими такового.

Основными программными техниками защиты являются:

– фильтрация спецсимволов – превращение в пользовательских данных интерпретируемых как служебные символов в интерпретируемые как текст, например, полным запретом на выполнение запроса при наличии таких символов или экранированием спецсимволов (применение другого служебного спецсимвола – чаще всего это обратный слэш / – чтобы символ интерпретировался именно как текст);

– фильтрация по типу данных – если заранее известно, данные какого формата должен вводить пользователь, следует ограничивать этот параметр до этих рамок – например, если известно, что пароль не может быть длиннее 10 символов и не должен содержать определённых символов, следует при превышении длины или наличии запрещённых символов в пароле отклонять запрос;

– ограничение возвращаемой информации – пользователь должен получать только абсолютный минимум необходимой информации в качестве ответа, и никогда не должен получать необработанного ответа от СУБД. Это включает в себя обязательный отказ от возвращения сообщения об ошибке в той форме, в которой его предоставляет СУБД, а также невозврат вообще каких бы то ни было данных при подозрении в том, что происходит попытка атаки;

– применение параметризованных запросов – многие СУБД и языки программирования, в том числе SQL Server, поддерживают использование заранее подготовленных процедур при программировании, которые, учитывая свойства СУБД, обрабатывают передаваемый пользователем текст и интерпретируют его исключительно как параметр, но не как код. Корректность данного метода напрямую зависит от разработчика кода параметризированных запросов. Для языка программирования C#, например, существуют коллекции SqlCommandBuilder, SqlDataAdapter и SqlParameter.

– учёт размера буфера данных – более актуально при использовании низкоуровневого программирования или процедур в СУБД. Необходимо проверять передаваемые пользователем данные на предмет того, чтобы они не превысили определённый при создании процедуры допустимый размер переменных; если это произойдёт, лишние данные будут усечены и добавлены в другую часть кода, которая не предназначена для этой переменной.

Главный принцип при программировании приложений таков – никогда и ни на каком этапе обработки данных нельзя доверять данным, передаваемым от пользователя.

В качестве заключения к теоретической части хотелось бы добавить, что защита от SQL-инъекций, как и всякий процесс защиты, должен быть комплексным и многоуровневым. Не существует ни одного "волшебного" средства, которое бы разом закрыло проблему, и всегда необходимо придерживаться концепции многослойной обороны – поскольку даже если злоумышленнику удастся преодолеть один слой (одно средство), за ним будут находиться другие.

Ход работы

В рамках данной лабораторной работы рассмотрим три наиболее основных вида SQL-инъекций: считывание данных (числовых и строковых), изменение данных, слепая инъекция. Поскольку непосредственного средства защиты от SQL-инъекций нет, необходимо программировать приложение и составлять запросы к СУБД с учётом возможности таковых инъекций, рассмотрим применение инъекций, чтобы было понятно, уязвимости какого формата эксплуатируются. Применение инъекций будет рассмотрено на примере виртуальной машины OWASP Broken Web Applications Project [https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project]. Данная виртуальная машина представляет собой компиляцию из множества преднамеренно уязвимых веб-приложений, предназначение которой заключается в обучении основам построения защищённых веб-приложений на ос-

новании «анти-примера». Все приложения работают в рамках виртуальной машины, которая не имеет доступа к содержимому хоста. Доступ к ней осуществляется по веб-интерфейсу со стороны хоста. Приложения включают в себя самый различный спектр уязвимостей, от XSS (кросс-сайт скриптинга) и DOS-атак до запуска файлов и собственно SQL-инъекций. Приложения подразделены на категории: обучающие приложения, предназначенные для непосредственного обучения; реалистичные приложения с уязвимостями, но без подсказок, для проверки умения искать уязвимости; устаревшие версии реально существующих приложений с известными уязвимостями; приложения для тестирования инструментов (автоматизированных поисковиков уязвимостей); демонстрирующие небольшой аспект приложения и демонстрационное приложение OWASP.

В рамках данной лабораторной работы рассмотрим только SQL-инъекции. В рамках хода работы рассмотрим тренировочное приложение OWASP WebGoat. Запустите виртуальную машину, используя VMware. Подождите её запуска. Машина работает в режиме командной строки. Когда машина будет запущена, отобразится приветственное сообщение («Welcome to the OWASP Broken Web Apps VM»). Из этого сообщения необходимо взять IP-адрес машины. IP-адрес следует ввести в строку адреса любого браузера на хосте (следует обратить внимание на то, что в браузере желательно включить java-скрипты, если они отключены). Откроется основное окно веб-приложений (Рисунок 7.1). Выберите приложение WebGoat. Приложение запросит логин/пароль, введите user/user (в дальнейшем для всех остальных приложений, где требуется пара логин/пароль, используется аналогичная пара для получения изначального доступа к нему). Для начала работы нажмите на кнопку Start WebGoat. Откроется основное окно приложения (Рисунок 7.2).

Рисунок 7.1 – Основное окно выбора веб-приложения

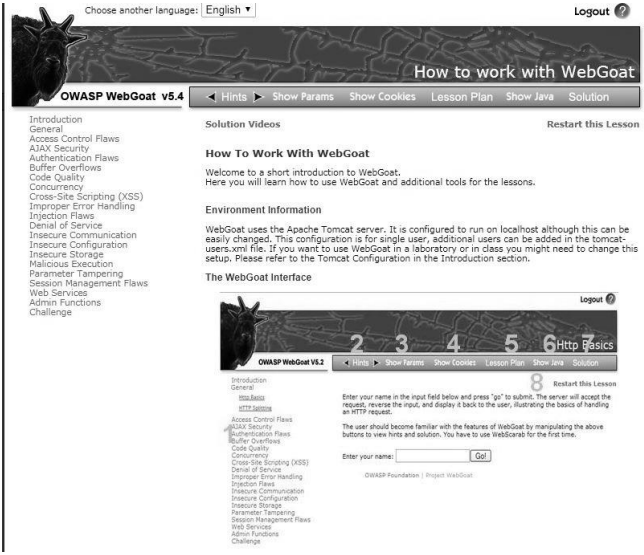


Рисунок 7.2 – Главное окно WebGoat

WebGoat работает вокруг понятия «урока». Уроки подразделяются на категории, каждая категория относится к одному из видов атак, каждый урок демонстрирует конкретную атаку. SQL-инъекции располагаются в категории Injection Flaws. В рамках хода работы рассмотрим следующие уроки: Numeric SQL Injection, String SQL Injection, Modify Data with SQL Injection, Blind String SQL Injection (числовая инъекция, строковая инъекция, модификация данных с помощью инъекции, слепая инъекция).

WebGoat поддерживает достаточно гибкую систему обучения, в рамках каждого урока описывается вкратце уязвимость, цель и имеется возможность посмотреть подсказки, используя кнопку Hints, и даже узнать непосредственное решение с помощью кнопки Solution. На основании этой информации, а также теоретического материала, данного в лабораторной работе, попробуйте решить уроки самостоятельно, прежде чем обращаться к нижеописанному ходу работы и/или непосредственному решению.

Numeric SQL Injection.

Данный вид инъекции основан на вводе вредоносного кода вместо значения, которое по задумке разработчиков должно быть числом. Синтаксической особенностью здесь является то, что число не закрывается кавычками, соответственно, продолжать код можно непосредственно. По заданию урока необходимо из запроса о погоде одной из четырёх станций получить данные обо всех станциях за раз. Запрос основан на номере станции. Чтобы получить

данные обо всех станциях, достаточно дополнить запрос логическим заключением, которое всегда истинно, например, 1=1. Таким образом, итоговые пользовательские данные примут вид 1 or 1=1. Однако на данной странице нет окна ввода, есть только окно выбора станции. Чтобы ввести инъекцию, следует отредактировать код страницы. Для этого следует открыть панель редактирования (F12 в браузере Firefox), где найти нужную строчку, введя в окно поиска ближайшее к выбору станции слово, и заменить текст в рамках одного из выборов на нужную инъекцию. Этот процесс продемонстрирован на Рисунках 7.3 и 7.4 (браузер Firefox, однако другие браузеры должны поддерживать подобный функционал).

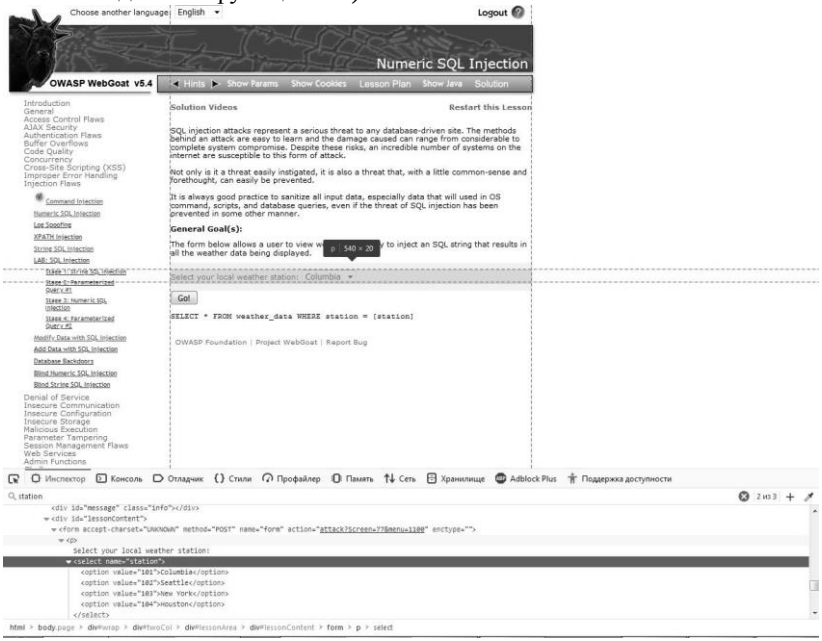


Рисунок 7.3 – Выбор нужной строки

Рисунок 7.4 – Ввод инъекции

В результате данной инъекции будет отправлен запрос SELECT * FROM weather_data WHERE station = 1 or 1=1. Условие в WHERE всегда истинно, соответственно, будут получены все данные о погоде во всех станциях, включая те, которых нет в окне выбора станции.

Таким образом, и из выбора только нескольких вариантов злоумышленник может определить момент добавления непосредственного кода и внедрить туда свою инъекцию.

String SQL Injection.

Данный вид инъекции аналогичен числовой, только в окне пользовательских данных вводится строка. Синтаксическое отличие тут в том, что строка в коде запроса заключается в кавычки, соответственно, для дополнения запроса необходимо избавиться от них с правой стороны. Для этого используется комментирование (текст за символами, помечающими комментарий, игнорируется отладчиком), которое в данном случае реализуется сочетанием двух дефисов (--). Согласно заданию урока, необходимо получить из таблицы с данными о кредитных картах все значения. Для этого следует в окно пользовательских данных ввести код 1' or 1=1; -- В результате этого получится запрос SELECT * FROM user_data WHERE last_name = '1' or 1=1; --' – кавычка за пометкой о комментарии будет проигнорирована, и запрос окажется синтаксически верным. Результат инъекции показан на Рисунке 7.5.

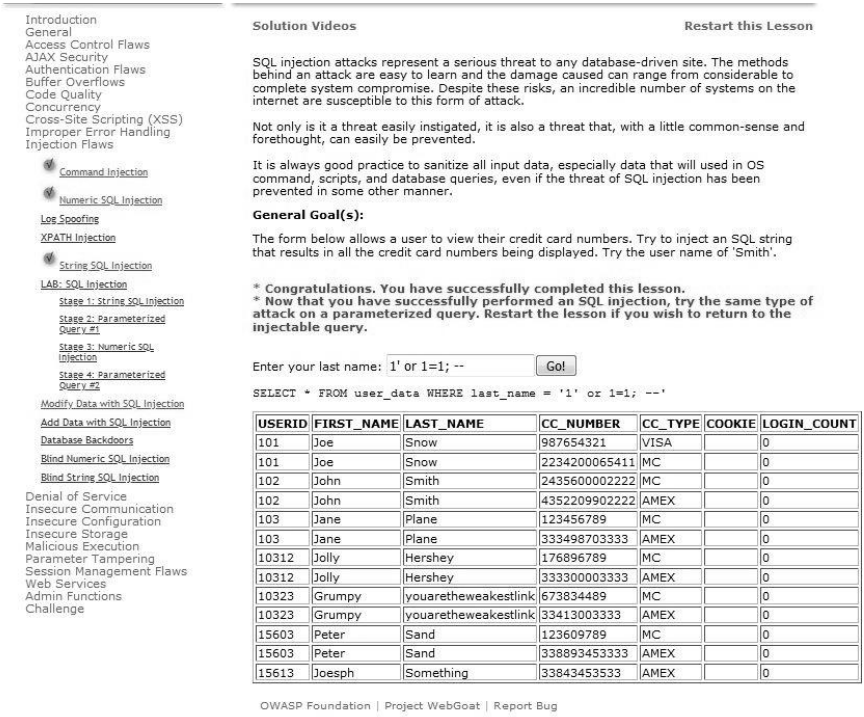


Рисунок 7.5 – Успешная строковая инъекция

Таким образом, даже при синтаксических ограничениях злоумышленник может подобрать такой вид инъекции, который по-прежнему оставит запрос синтаксически верным.

Modify Data with SQL Injection.

Данный вид инъекции строится вокруг изменения данных в процессе запроса к данным. Принципиальное отличие этой инъекции от ранее рассмотренных в том, что к уже имеющемуся запросу добавляется новый. Согласно заданию урока, необходимо изменить зарплату пользователя jsmith в окне, которое по задумке позволяет только считывать данные о зарплатах. Для того, чтобы это сделать, следует ввести в окне пользовательских данных код:

1'; UPDATE salaries SET SALARY = 99999 WHERE USERID = 'jsmith';
--

В результате данной инъекции зарплата будет изменена, что впоследствии можно проверить, указав имя пользователя (Рисунок 7.6).

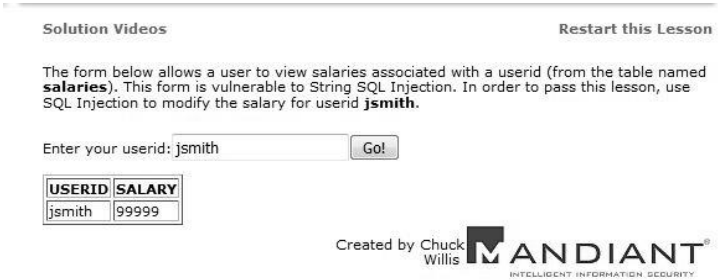


Рисунок 7.6 – Успешная инъекция с изменением данных

Таким образом, злоумышленник при использовании инъекций может не ограничиваться только тем запросом, к которому вводятся данные.

Blind String SQL Injection.

Данный вид инъекции принципиально отличается от остальных инъекций тем, что в результате запроса пользователь видит только бинарный ответ (да/нет). Простейшим примером такого рода запроса является окно аутентификации: при успешной аутентификации пользователь будет авторизован ("да"), при неуспешной будет возвращён соответствующий ответ ("нет"). Согласно заданию урока, необходимо определить значение столбца pin в таблице pins для строки, в которой столбец cc_number имеет значение 1111222233334444, при том, что изначальный функционал позволяет только проверять, является ли аккаунт с указанным номером валидным (бинарный ответ). Известно, что столбец pin является целым числом. Для того, чтобы это определить, необходимо поменять задаваемый вопрос, указывая заранее достоверно известный номер (например, 101) и дополняя вопрос уточнением того, в каких границах находится значение столбца. Например (с чего можно начать):

101 AND ((SELECT pin FROM pins WHERE cc_number='1111222233334444') > 10000);

Будет возвращено, что номер не валидный. Это значит, что наше искомое значение меньше 10000. Таким образом меняя границы вопроса, определите, каким является это число, и введите его в окно запроса, чтобы урок был завершён.

Таким образом, даже когда злоумышленник ограничен в получаемой им информации, он по-прежнему может найти способ получить нужную ему информацию посредством инъекции.

Контрольные вопросы

1) Дайте определение понятию SQL-инъекции;
2) Опишите отличительные признаки слепой и двойной слепой инъекции от обычной;
3) Каким образом можно защититься от SQL-инъекции?
4) Перечислите несколько (3-4) программных техники, с помощью которых можно ограничить некоторые виды инъекций;
5) Чего злоумышленник может добиться посредством SQL-инъекции? За какие границы он не может выйти посредством одних только инъекций?
6) Приведите конкретный пример SQL-инъекции на следующем запросе: SELECT * FROM salaries WHERE user_name = '[userdata]'; – в запросе строка [userdata] заменяется на пользовательские данные.

Задание

Выполните Ход работы. Найдите в приложении согласно варианту три вида инъекций: считывание данных, изменение данных, слепая инъекция. Составьте отчёт по проделанной работе, в котором обязательно продемонстрируйте применение обнаруженных инъекций и конкретный результат их применения. Продемонстрируйте работу преподавателю.

Литература

1. Шустова Л.И., Тараканов О.В. Базы данных: учеб. – М.: НИЦ ИНФРА-М, 2016. – 304 с.

2. Култыгин О.П. Администрирование баз данных СУБД MS SQL Server [Электронный ресурс]: учеб. пособие. – М.: МФПА, 2012. – 232 с.:

3. Агальцов В.П. Базы данных. В 2-х кн. – Кн. 1: Локальные базы данных: учеб. – 2-е изд., перераб. – М.: ИД ФОРУМ: ИНФРА-М, 2012. – 352 с.

Учебное издание

Константин Сергеевич Сарин,
Александр Сергеевич Киселев

БЕЗОПАСНОСТЬ СИСТЕМ БАЗ ДАННЫХ

Лабораторный практикум
для студентов специальностей и направлений
10.03.01 – «Информационная безопасность»,
10.05.02 – «Информационная безопасность телекоммуникационных систем»,
10.05.03 – «Информационная безопасность автоматизированных систем»,
10.05.04 – «Информационно-аналитические системы безопасности»

Верстка – В.М. Бочкаревой
Текст дан в авторской редакции, без корректуры

Издательство «В-Спектр»
Подписано к печати 10.01.2020.
Формат $60\times84^{1}/_{16}$. Печать трафаретная.
Печ. л. 2,75. Тираж 100 экз. Заказ 19.

Тираж отпечатан ИП В.М. Бочкаревой
ИНН 701701817754
634055, г. Томск, пр. Академический, 13-24, тел. 89050899240
E-mail: bvm@sibmail.com