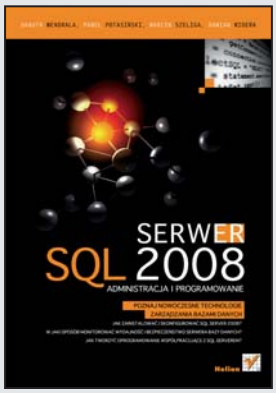
Serwer SQL 2008. Administracja i programowanie



Autor: Danuta Mendrala, Paweł Potasiński, Marcin Szeliga, Damian Widera
ISBN: 978-83-246-2033-3
Format: 158x235, stron: 488

Poznaj nowoczesne technologie zarządzania bazami danych

- Jak zainstalować i skonfigurować SQL Server 2008?
- W jaki sposób monitorować wydajność i bezpieczeństwo serwera bazy danych?
- Jak tworzyć oprogramowanie współpracujące z SQL Serverem?

System zarządzania bazami danych MS SQL Server zyskał sobie zasłużoną renomę wśród projektantów baz danych i programistów. Stabilna i wydajna platforma, dostępna w kilku edycjach, różniących się możliwościami i zastosowaniami, wykorzystywana jest w aplikacjach biznesowych, portalach internetowych i aplikacjach mobilnych. Jej najnowsza wersja, oznaczona symbolem 2008, to potężna platforma do zarządzania danymi, umożliwiająca nie tylko przechowywanie ich w tabelach, ale także przetwarzanie, analizowanie, udostępnianie i zabezpieczanie. Pozwala ona także na składowanie danych nierelacyjnych – obiektów binarnych i danych geograficznych.

Książka „Serwer SQL 2008. Administracja i programowanie" zawiera szczegółowe omówienie możliwości najnowszej wersji tej platformy bazodanowej. Zarówno administratorzy, jak i programiści znajdą w niej wiele przydatnych informacji. Opisano w niej poszczególne edycje SQL Servera 2008, sposób ich instalacji, konfiguracji i administrowania, techniki optymalizacji wydajności oraz zabezpieczania i analizowania danych. Dzięki niej dowiesz się, jakie nowe funkcje wprowadzono w wersji 2008, jak korzystać z języka T-SQL i przechowywać dane nierelacyjne. Przeczytasz także o integracji z platformą CLR oraz korzystaniu z technologii SOA.

- Instalacja SQL Servera 2008
- Administrowanie serwerem bazy danych
- Optymalizacja dostępności i wydajności
- Nowe funkcje SQL Servera 2008
- Monitorowania pracy serwera
- Zabezpieczanie danych
- Programowanie w T-SQL
- Nierelacyjne typy danych
- Korzystanie z dokumentów XML
- Praca z SQL Server Compact Edition

Wykorzystaj w praktyce możliwości najnowszej wersji platformy bazodanowej SQL Server

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Rozdział 7.

Bezpieczeństwo

Bezpieczeństwo serwera SQL zależy w tym samym stopniu od administratorów co od programistów baz danych, a więc ten rozdział równie dobrze mógłby być pierwszym rozdziałem drugiej części książki. O jego lokalizacji zadecydowało to, że dwa najważniejsze nowe zabezpieczenia serwera SQL 2008 przeznaczone są dla administratorów.

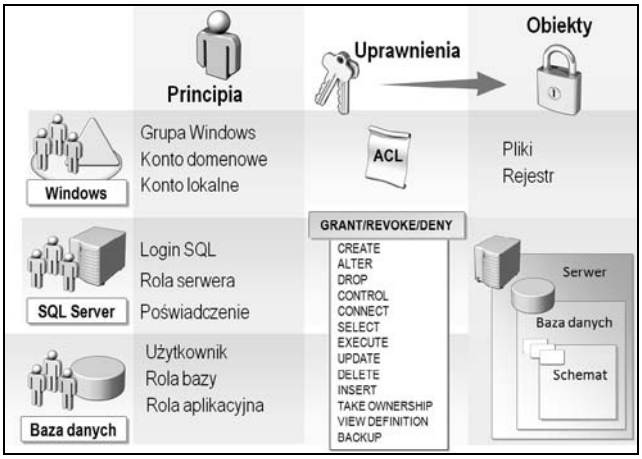
W bieżącym rozdziale przedstawiliśmy, oprócz mechanizmów szyfrowania baz danych TDE (ang. *Transparent Data Encryption*) i monitorowania wszystkich zdarzeń (ang. *All Action Audit*), model bezpieczeństwa serwera SQL i jego funkcje kryptograficzne.

Model bezpieczeństwa serwera SQL 2008

Serwer SQL 2008 będzie co najwyżej tak bezpieczny jak system operacyjny, w środowisku którego działa. Jeżeli uruchomione na nim będą niepotrzebne usługi czy pochodzące z niezaufanych źródeł programy, a dostęp do komputera (zarówno lokalny, jak i sieciowy) nie zostanie ograniczony, zabezpieczenia na poziomie serwera SQL będą nieskuteczne. **Choć serwer SQL 2008 umożliwia skuteczne zabezpieczenie baz danych, to jest ściśle zintegrowany z systemem Windows** (rysunek 7.1).

Pierwsza kolumna zawiera reprezentujące użytkowników principia. Ich cechą wspólną jest możliwość potwierdzania swojej tożsamości — **serwer SQL 2008 nie pozwala na nawiązywanie anonimowych sesji klienckich**. Principia występują na trzech poziomach:

1. Na poziomie systemu operacyjnego dysponujemy kontami użytkowników i grupami użytkowników systemu Windows. Chociaż serwer SQL 2008 może uwierzytelniać użytkowników lokalnego systemu Windows, został zaprojektowany z myślą o kontach domenowych.



Rysunek 7.1. *Rozbudowany model bezpieczeństwa serwera SQL 2008*

2. Na poziomie instancji serwera SQL mamy do dyspozycji loginy, stałe role serwera i dodatkowe poświadczenia. Pozwalają one uwierzytelniać użytkowników (również niemających kont w systemie Windows), łączyć ich w role i nadawać uprawnienia do zewnętrznych zasobów, np. plików.

3. Na poziomie bazy danych dysponujemy kontami użytkowników i rolami baz danych. Służą one do nadawania uprawnień w wybranej bazie danych, a jeden login serwera może być połączony z kontami użytkowników w wielu bazach[1]. Użytkownik, który uwierzytelnił się na podstawie loginu niepowiązanego w danej bazie z żadnym kontem użytkownika, uzyska do niej dostęp, tylko jeżeli włączone jest w niej konto gościa.

W trzeciej kolumnie pokazana została hierarchia obiektów, do których dostęp kontrolują listy ACL (ang. *Access Control List*). **W większości przypadków obiekty podrzędne dziedziczą uprawnienia po obiektach nadrzędnych**:

1. Zagadnienie zabezpieczenia obiektów systemu Windows, takich jak pliki baz danych czy klucze rejestru, wykracza poza zakres książki.

2. Na poziomie instancji serwera SQL znajdują się m.in. loginy, role serwera, urządzenia kopii zapasowych, wyzwalacze DDL i logowania, główny klucz usługi SMK oraz powiązane serwery. Ponieważ loginy są zarówno principiami, jak i obiektami, możemy nadawać uprawnienia do loginów, np. pozwolić wskazanym osobom na ich blokowanie czy zmienianie haseł.

3. Na poziomie bazy danych znajdują się schematy, typy danych, konta użytkowników, role bazy danych (do których domyślnie należą niektóre loginy i role serwera), certyfikaty, klucze kryptograficzne, wyzwalacze DDL, biblioteki kodu zarządzanego i schematy XML.

4. Na poziomie schematów znajdują się pozostałe obiekty bazodanowe, w tym tabele, widoki, procedury, funkcje użytkownika, wyzwalacze DML, synonimy i kolejki usługi Service Broker.

[1] Domyślnie nazwa konta użytkownika jest taka sama jak nazwa loginu.

Pomiędzy principiami a obiektami znajdują się uprawnienia, które mogą być nadane principiom do obiektów (ang. *grant*), jawnie odebrane (ang. *deny*) lub nieokreślone (ang. *revoke*). Serwer SQL autoryzuje wszystkie operacje użytkowników, a więc przed ich wykonaniem sprawdza, czy dane principia mają wystarczające do ich wykonania uprawnienia — jeżeli autoryzacje się nie powiodą, operacje zostają przerwane.

Żeby efektywnie zarządzać uprawnieniami, nie nadaje się ich poszczególnym użytkownikom, a całym rolom. Ponieważ uprawnienia się kumulują, principium należące do dwóch ról będzie miało uprawnienia ich obu. Wyjątkiem od tej reguły jest jawne odebranie uprawnienia — w takim przypadku principium nie wykona zabronionej operacji niezależnie od tego, jakie uprawnienia mają nadane role, do których należy.

Uwierzytelnianie

Każdy użytkownik, zanim nawiąże sesję z serwerem, musi być uwierzytelniony. Tożsamość użytkowników może być potwierdzana przez:

1. System operacyjny (tryb *Windows Authentication*). Zdecydowanie bezpieczniejszy i jeśli użytkownicy serwera SQL mają konta w domenie AD, pozwalający niewielkim nakładem pracy uzyskać funkcjonalny model zarządzania użytkownikami. W tym trybie serwer SQL w ogóle nie sprawdza tożsamości użytkowników, w pełni ufając systemowi Windows. Tryb Windows jest trybem domyślnym, co oznacza, że chociaż można tworzyć loginy SQL, to próba zalogowania się za ich pomocą do serwera, nawet po podaniu prawidłowego hasła, skończy się błędem.
2. Serwer SQL (tryb *SQL Server and Windows Authentication*) — w tym trybie możliwe jest uwierzytelnianie na podstawie konta systemu Windows i utworzonego dla niego loginu oraz poprzez podanie loginu SQL i hasła. Podane przez użytkownika login i hasło są przed wysłaniem do serwera SQL szyfrowane jego certyfikatem, a następnie porównywane z zapisanymi w bazie master loginami i skrótami haseł.

W trybie Windows to, jaki zostanie użyty protokół uwierzytelniania (Kerberos czy NTLM), zależy od konfiguracji systemu Windows, serwera SQL lub aplikacji klienckiej. Żeby serwer SQL mógł użyć znacznie bezpieczniejszego protokołu Kerberos:

1. Serwer SQL i komputer kliencki muszą znajdować się w tej samej domenie lub zaufanych domenach.
2. Administrator musi zarejestrować w domenie nazwę SPN serwera SQL 2008[2] albo programista musi wskazać w aplikacji klienckiej odpowiednie konto systemowe. Wymaga to użycia w ciągu połączenia nazwy UPN (ang. *Unviersal Principal Name*), np. w formacie `nazwa domeny\konto serwera SQL`.

[2] Jeżeli serwer działa z uprawnieniami konta *Local System*, rejestracja będzie przeprowadzona automatycznie, w innym przypadku należy skorzystać z wchodzącego w skład pakietu Resource Kit narzędzia SetSPN.

Serwer SQL 2008 może używać protokołu Kerberos z wszystkimi protokołami sieciowymi. W poprzednich wersjach Kerberos działał wyłącznie z protokołem TCP/IP.

Żeby zmienić tryb uwierzytelniania:

1. Połącz się za pomocą konsoli MSSM z serwerem SQL jako jego administrator.
2. W oknie eksploratora obiektów kliknij nazwę serwera prawym przyciskiem myszy i z menu kontekstowego wybierz *Properties*.
3. Przejdź na zakładkę *Security* i wybierz pole wyboru *SQL Server and Windows Authentication Mode*.
4. Kliknij *OK* i uruchom ponownie serwer (można to zrobić, klikając jego nazwę prawym przyciskiem myszy i wybierając zadanie *Restart*).

Loginy

Tworząc login, możemy wskazać domyślną bazę danych — z tą bazą użytkownik połączy się automatycznie (o ile będzie w niej miał konto) — oraz domyślny język. Zmiana języka ma sens tylko wtedy, kiedy planujemy przetłumaczyć na ten język systemowe komunikaty błędów. W takim przypadku użytkownik po wystąpieniu błędu zobaczy jego przetłumaczoną wersję. Login możemy utworzyć dla:

1. osoby, która nie ma konta w systemie Windows;
2. pojedynczego konta użytkownika systemu Windows;
3. grupy użytkowników systemu Windows.

Szczególnie wygodna i funkcjonalna jest trzecia opcja. **Po utworzeniu loginów dla grupy użytkowników osoba, która zmieni stanowisko i tym samym członkowstwo w grupach Windows, automatycznie będzie miała zmienione uprawnienia w serwerze SQL**.

Zarządzać loginami, w tym zakładać nowe, zmieniać hasła, przypisywać je do ról serwera oraz łączyć z kontami użytkowników baz danych możemy za pomocą konsoli SSMS — po rozwinięciu sekcji *Security/Logins* w oknie eksploratora obiektów wyświetlone zostaną wszystkie loginy. Klikając dowolny z nich prawym przyciskiem myszy, wyświetlimy menu kontekstowe pozwalające m.in. pokazać okno szczegółów wybranego loginu.

Wykonując poniższą instrukcję, utworzymy nowy login SQL:

```
CREATE LOGIN Szelor
WITH PASSWORD= 'CoZ@Dzien!' , DEFAULT_DATABASE=AdventureWorks2008, CHECK_EXPIRATION=ON,
➥CHECK_POLICY=ON;
```

Zasady konta

Powszechnie używane do konfigurowania systemów Windows zasady grupy (ang. *Group Policy*) zawierają m.in. zasady haseł i zasady blokady konta. Jeżeli serwer SQL 2008 działa w środowisku systemów Windows Server 2003, Windows Vista lub późniejszych, to **obowiązujące na tym komputerze zasady konta będą mogły być również stosowane do loginów SQL**[3].

Zasady konta podzielone są na dwie grupy.

1. Zasady haseł, w ramach których należy:
   a) Włączyć regułę *Hasło musi spełniać wymagania co do złożoności*.
   b) Określić maksymalny okres ważności hasła, czyli liczbę dni, po których będzie ono musiało być zmienione; hasła powinny być zmieniane nie rzadziej niż co trzy miesiące.
   c) Określić minimalny, nawet jednodniowy okres ważności hasła; w ten sposób uniemożliwimy użytkownikom szybką zmianę hasła tyle razy, żeby z powrotem mogli używać poprzedniego.
   d) Wymusić tworzenie długiej, liczącej nawet 20 pozycji historii haseł — ta zasada w połączeniu z poprzednią zmusi użytkowników do faktycznego zmieniania haseł.
2. Zasady blokady konta, w ramach których należy:
   a) Określić przynajmniej kilkunastominutowy czas, na który konto zostanie automatycznie zablokowane.
   b) Ustawić wysoki próg blokady konta; ryzyko, że złożone hasło zostanie odgadnięte w pięciu i w piętnastu próbach, jest praktycznie takie samo — ustawiając zbyt niski próg blokady, tylko utrudnimy sobie pracę.

Zablokowane loginy można odblokować za pomocą instrukcji `ALTER LOGIN … UNLOCK`.

Autoryzacja

Uwierzytelniony podczas połączenia z serwerem SQL użytkownik może wykonać na nim tylko te operacje, do których zostały mu nadane uprawnienia. Uprawnienia dzielą się na nadawane do konkretnych obiektów (np. uprawnienie do odczytywania danych z tabeli `Person.Person`) oraz prawa do wykonywania wskazanych instrukcji (np. prawo do tworzenia kopii zapasowych). Biorąc pod uwagę liczbę obiektów bazodanowych i kont użytkowników, zarządzanie uprawnieniami na poziomie poszczególnych kont użytkowników jest nieefektywne i niebezpieczne — **administrator nie będzie w stanie**

[3] Wyłączyć stosowanie zasad konta możemy za pomocą klauzul `CHECK_EXPIRATION` i `CHECK_POLICY` instrukcji `ALTER LOGIN`.

na bieżąco nadawać i odbierać uprawnień i w końcu przyzna wszystkim użytkownikom pełne uprawnienia. Rozwiązanie tego problemu polega na wykorzystaniu hierarchii obiektów oraz ról serwera i baz danych.

Role serwera

Tworzenie, usuwanie lub zmiana uprawnień ról serwera są niemożliwe. Możemy jedynie do tych ról dodawać lub usuwać loginy i w ten sposób uprościć nadawanie uprawnień na poziomie serwera SQL. Warto się zastanowić, zanim dodamy jakiś login do uprzywilejowanej grupy — **w serwerze SQL 2008 obowiązuje zasada, że principium należące do jakiejś roli może dopisać do niej kolejne osoby**. Listę predefiniowanych ról serwera zwraca procedura `sp_helpsrvrole`, listę uprawnień każdej z ról — procedura `sp_srvrolepermission`:

1. `bulkadmin` — członkowie tej roli mogą wykonywać instrukcję `BULK INSERT`.
2. `dbcreator` — członkowie tej roli mogą tworzyć, usuwać, modyfikować i odtwarzać kopie baz danych.
3. `diskadmin` — członkowie tej roli mogą konfigurować pliki baz danych.
4. `processadmin` — członkowie tej roli mogą przerwać (wykonując instrukcję `KILL`) sesję dowolnego użytkownika serwera.
5. `public` — do tej roli automatycznie należą wszyscy użytkownicy serwera; rola `public` jest powiązana z kontem gościa w każdej bazie danych.
6. `securityadmin` — członkowie tej roli mogą zarządzać loginami (w tym resetować hasła) oraz uprawnieniami na poziomie serwera i baz danych.
7. `serveradmin` — członkowie tej roli mogą konfigurować, uruchamiać i zatrzymywać serwer SQL.
8. `setupadmin` — członkowie tej roli mogą dodawać, zmieniać i usuwać powiązane serwery.
9. `sysadmin` — członkowie tej roli mogą wykonywać dowolne operacje. Domyślnie do tej roli należą:
   a) login *sa*,
   b) wbudowana grupa Windows *BUILTIN\Administrators*.

Do ról serwera można dodawać loginy lub usuwać je z nich z poziomu konsoli SSMS, poprzez okno właściwości loginu albo okno właściwości roli serwera.

Dodatkowe poświadczenia

Dodatkowymi poświadczeniami tożsamości mogą się posłużyć osoby niemające kont w systemie Windows, żeby uzyskać dostęp do zasobów systemowych. Na przykład, jeżeli osoba uwierzytelniona na podstawie loginu SQL chce wykonać procedurę, która odczytuje pliki z dysku NTFS, serwer SQL wykona tę operacje z uprawnieniami wskazanego użytkownika systemu Windows. Konfigurując dodatkowe poświadczenia, należy:

1. Utworzyć konto w domenie i nadać mu odpowiednie uprawnienie (np. uprawnienie do odczytu z danego folderu[4]).
2. Dodać poświadczenie, wskazując utworzone wcześniej konto i chroniące je hasło[5]:

```
CREATE CREDENTIAL AlterEgo
WITH IDENTITY = 'KATMAI\SqlCred',
SECRET = 'BezpIecznEH@sl0jESTDl*^%ie';
```

3. Powiązać utworzony wcześniej login z dodatkowym poświadczeniem (z jednym poświadczeniem można powiązać wiele loginów, ale ten sam login może być powiązany tylko z jednym poświadczeniem):

```
ALTER LOGIN Szelor
WITH CREDENTIAL = AlterEgo;
```

Konta użytkowników

Łącząc się z serwerem SQL, nawiązujemy sesję z bazą danych — domyślnie jest to systemowa baza master, ale docelową bazę można wskazać w zakładce *Connection Properties* okienka logowania konsoli SSMS lub zdefiniować ją jako właściwość loginu[6]. Jeżeli w docelowej bazie danych nie będzie powiązanego z loginem konta użytkownika, połączymy się z nią jako gość (konto gościa jest odblokowane w systemowych bazach danych) albo zgłoszony zostanie błąd 4064: *Cannot open user default database. Login failed*.

Tworząc konto użytkownika, czy to za pomocą konsoli SSMS, czy wykonując instrukcję `CREATE USER`, należy wskazać login lub certyfikat, z którym zostanie ono powiązane:

```
USE AdventureWorks2008
CREATE USER Szelor
FOR LOGIN Szelor;
```

Wywołując funkcję `USER_NAME()`, możemy sprawdzić nazwę użytkownika, pod którą jesteśmy połączeni z bazą danych:

```
USE master
SELECT USER_NAME();
USE AdventureWorks2008
SELECT USER_NAME();
--------------------------------------
guest
Szelor
```

[4] Hasło tego konta nie powinno wygasać i nie może wymagać zmiany przy pierwszym logowaniu.

[5] Jeżeli podamy błędne hasło, poświadczenie zostanie utworzone, ale próba posłużenia się nim skończy się błędem.

[6] Wybranie dla loginu domyślnej bazy danych nie tworzy w niej powiązanego z nim konta użytkownika.

Role bazy danych

Każda baza danych zawiera następujące predefiniowane, niemożliwe do usunięcia oraz skonfigurowania role:

1. `db_accessadmin` — jej członkowie mogą kontrolować dostęp innych użytkowników do bazy danych.
2. `db_backupoperator` — należący do niej użytkownicy mogą tworzyć kopie zapasowe.
3. `db_datareader` — jej członkowie mogą odczytywać zawartość tabel bazy danych, z wyjątkiem tabel systemowych.
4. `db_datawriter` — jej członkowie mogą modyfikować, usuwać i dodawać dane do tabel bazy danych, z wyjątkiem tabel systemowych.
5. `db_ddladmin` — jej członkowie mogą wykonywać wszystkie instrukcje DDL.
6. `db_denydatareader` — jej członkowie nie mogą odczytywać zawartości tabel bazy danych.
7. `db_denydatawriter` — należący do niej użytkownicy nie mogą modyfikować, usuwać i dodawać danych.
8. `db_owner` — jej członkowie mogą przeprowadzać dowolne zmiany w bazie danych, nawet ją usunąć.
9. `db_securityadmin` — jej członkowie mogą kontrolować członkostwo w innych rolach i zarządzać uprawnieniami użytkowników.
10. `public` — do tej roli automatycznie należą wszyscy użytkownicy bazy.

W przeciwieństwie do ról serwera, możliwe jest tworzenie własnych ról bazy danych. Najczęściej role bazy danych tworzy się, jeżeli używane są loginy SQL — taki login reprezentuje jednego użytkownika, natomiast login Windows może reprezentować całą grupę użytkowników. Dodatkowe role bazy danych początkowo nie mają nadanych żadnych uprawnień, a więc zapisanie do nich użytkowników nie ma wpływu na bezpieczeństwo serwera.

W bazach danych możemy też tworzyć role aplikacyjne — specjalny typ ról bazy danych, które muszą być programowo włączone. Niemożliwe jest przypisanie kont użytkowników do roli aplikacyjnej. Natomiast **po ich włączeniu poprzez podanie nazwy i hasła użytkownik traci wszystkie posiadane uprawnienia, zyskując w zamian uprawnienia nadane roli aplikacyjnej**.

Role aplikacyjne służą do identyfikacji programów klienckich (i przez nie powinny być włączane), a nie użytkowników. Jeżeli użytkownicy nie będą mieli nadanych uprawnień do wykonywania operacji (np. do odczytywania danych z tabeli), a wymagane uprawnienia będzie miała tylko rola aplikacyjna, wymusimy na użytkownikach korzystanie wyłącznie z określonego programu. Pokazuje to poniższy przykład:

1. Utworzymy w bazie AdventureWorks2008 rolę aplikacyjną:

```
USE AdventureWorks2008
CREATE APPLICATION ROLE testAppR
WITH DEFAULT_SCHEMA = HumanResources , PASSWORD = N'Hasl0DoZ@szyciaWaplikacji!';
```

2. Następnie nadamy tej roli uprawnienie do odczytywania danych ze schematu HumanResources:

```
GRANT SELECT ON SCHEMA::HumanResources TO testAppR;
```

3. I spróbujemy odczytać dane ze znajdującej się w tym schemacie tabeli Department:

   a) Najpierw jako użytkownik Szelor (to konto dopiero co założyliśmy, a więc nie ma ono żadnych uprawnień, z wyjątkiem uprawnień roli public):

```
EXECUTE AS USER = 'Szelor';
SELECT TOP 1 *
FROM HumanResources.Department;
---------------------------------------
Msg 229, Level 14, State 5, Line 2
The SELECT permission was denied on the object 'Department', database
↪'AdventureWorks2008', schema 'HumanResources'.
```

   b) A następnie jako członek włączonej na czas wykonywania zapytania roli aplikacyjnej:

```
DECLARE @cookie varbinary(8000);
EXEC sp_setapprole N'testAppR',N'Hasl0DoZ@szyciaWaplikacji!', @fCreateCookie
↪= true, @cookie = @cookie OUTPUT;
SELECT TOP 1 *
FROM HumanResources.Department;
EXEC sp_unsetapprole @cookie;

REVERT;
---------------------------------------
1 Engineering    Research and Development     1998-06-01 00:00:00.000
```

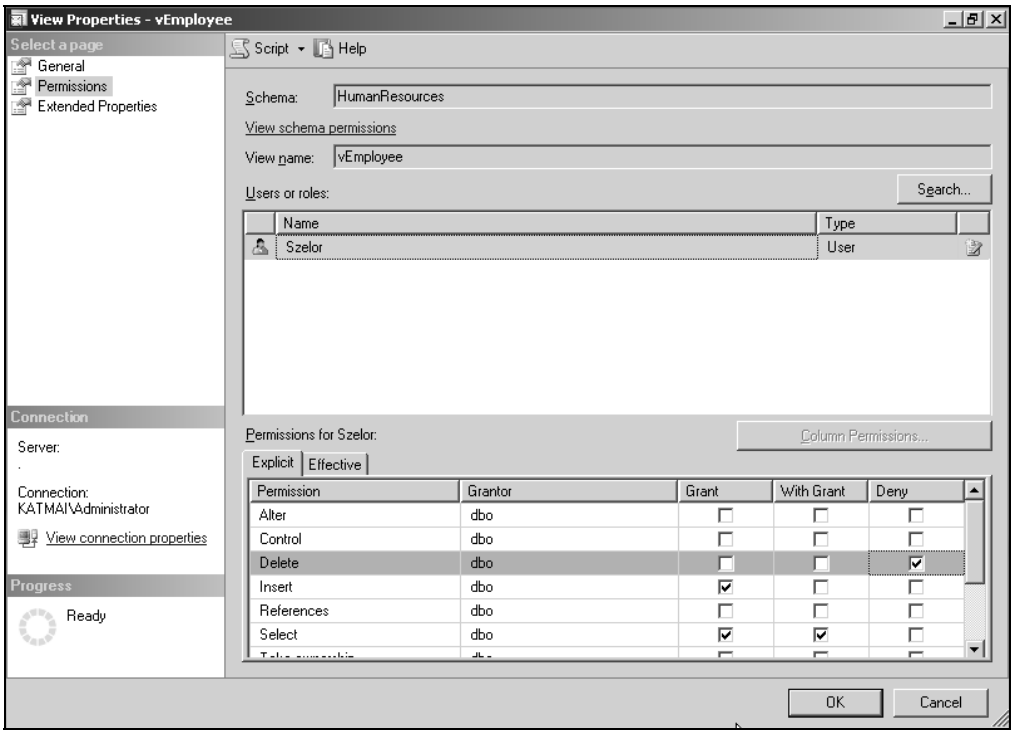
Uprawnienia

Lista wszystkich możliwych do nadania lub odebrania uprawnień liczy prawie 200 pozycji. Wyświetlić ją możemy, wykonując poniższe zapytanie:

```
SELECT *
FROM fn_builtin_permissions(default);
```

> Na specjalną uwagę zasługuje uprawnienie CONTROL — oznacza ono wszystkie możliwe dla danego obiektu uprawnienia. Innymi słowy, nadając komuś uprawnienie CONTROL do tabeli, pozwalamy mu na wykonanie na tej tabeli dowolnych operacji, w tym odczytywania i modyfikowania zawartości tabeli czy jej usunięcia.

Uprawnienie może być: nadane, nadane z możliwością przekazania, odebrane lub nieustalone. Zarządzać uprawnieniami możemy poprzez okno właściwości obiektu, okno właściwości principium lub wykonując instrukcje GRANT, DENY i REVOKE (rysunek 7.2).



Rysunek 7.2. *Zakładka Effective zawiera efektywne uprawnienia wybranego principium do obiektu — ponieważ uprawnienia mogą wynikać z członkostwa w różnych rolach, a niektóre z nich (np. uprawnienie CONTROL) są w rzeczywistości grupami uprawnień, znacznie ułatwia ona ocenę uprawnień użytkowników*

Brak uprawnienia, tak samo jak jego odebranie, oznacza, że użytkownik nie będzie mógł wykonać danej operacji. Różnica polega na tym, że brakujące uprawnienie może być odziedziczone (np. jeżeli zostało nadane roli, do której należy użytkownik), natomiast odebranie użytkownikowi uprawnienia oznacza, że nie wykona on danej operacji, niezależnie od ról, do których należy.

Schematy

Schematy to nie tylko przestrzenie nazewnicze. Schematy są obiektami, które zawierają obiekty innych typów (takie jak tabele lub procedury), i do nich także możemy nadawać uprawnienia. W ten sposób znacznie uprościmy czynności administracyjne — zamiast zezwalać użytkownikom na odczyt danych z poszczególnych tabel, możemy zezwolić im na odczyt danych z całego schematu. Pokazuje to poniższy przykład:

1. Nadajmy użytkownikowi `Szelor` uprawnienie do odczytu i wstawiania wierszy do całego schematu `HumanResources`:

```
GRANT SELECT, INSERT
ON SCHEMA ::HumanResources
TO Szelor;
```

2. Odbierzmy mu jednak uprawnienie do odczytu jednego z widoków tego schematu:

```
DENY SELECT
ON HumanResources.vJobCandidate
TO Szelor;
```

3. Sprawdźmy, czy ten użytkownik będzie mógł odczytać dane z jednej z tabel schematu `HumanResources`:

```
EXECUTE AS USER = 'Szelor'
SELECT TOP 1 *
FROM HumanResources.Shift;--------------------------------------
1 Day     07:00:00.0000000     15:00:00.0000000        1998-06-01 00:00:00.000
```

4. Oraz czy uda mu się odczytać dane poprzez widok `vJobCandidate`:

```
SELECT TOP 1 *
FROM HumanResources.vJobCandidate;

REVERT;
--------------------------------------
Msg 229, Level 14, State 5, Line 1
The SELECT permission was denied on the object 'vJobCandidate', database
➥'AdventureWorks2008', schema 'HumanResources'.
```

> Odwołując się do obiektów bazodanowych, należy zawsze poprzedzać ich nazwę nazwą schematu, w którym dany obiekt się znajduje.

Zmiana kontekstu wykonania instrukcji i łańcuchy własności

Domyślnie wszystkie instrukcje wykonywane są przez serwer SQL 2008 z uprawnieniami wywołującego je użytkownika. Można jednak to zmienić — w kilku poprzednich przykładach kontekst wykonania instrukcji był zmieniany za pomocą instrukcji `EXECUTE AS`. W podobny sposób można też zmienić kontekst wykonania modułu kodu (np. procedury składowanej). Takie rozwiązanie utrudnia jednak monitorowanie aktywności użytkowników i może umożliwić atak poszerzenia uprawnień[7].

Procedura składowana może być wykonana z uprawnieniami:

1. Użytkownika, który ją uruchomił (domyślna wartość `CALLER`).
2. Użytkownika, który ją utworzył lub jako ostatni zmodyfikował (`EXECUTE AS SELF`).
3. Użytkownika, który jest jej właścicielem (`EXECUTE AS OWNER`).
4. Wskazanego użytkownika (`EXECUTE AS nazwa użytkownika`).

[7] Obu tych wad pozbawione jest alternatywne, przedstawione w dalszej części rozdziału rozwiązanie polegające na cyfrowym podpisywaniu modułów kodu.

Pokazuje to poniższy przykład:

1. Po utworzeniu procedury nadajemy użytkownikowi `Szelor` uprawnienie do jej wykonania:

```
CREATE PROCEDURE Person.GetAddress
AS
SELECT TOP 1 *
FROM Person.Address;
GO
GRANT EXECUTE ON Person.GetAddress
TO Szelor;
```

2. Jeżeli użytkownik `Szelor` spróbuje bezpośrednio odwołać się do tabeli `Person.GetAddress`, serwer SQL zgłosi błąd:

```
EXECUTE AS LOGIN ='Szelor';
SELECT TOP 1 *
FROM Person.Address;
---------------------------------------
Msg 229, Level 14, State 5, Line 1
The SELECT permission was denied on the object 'Address', database
↪'AdventureWorks2008', schema 'Person'.
```

3. Jeżeli jednak odczyta on te same dane poprzez procedurę, serwer SQL nie zgłosi błędu:

```
EXEC Person.GetAddress;

REVERT;
---------------------------------------
1 1970 Napa Ct.  NULL   Bothell      79      98011    …
```

Zadziałał tu mechanizm łańcuchów własności obiektów — jeżeli obiekt, do którego bezpośrednio odwołuje się użytkownik (procedura `GetAddress`), jest własnością tego samego użytkownika (`dbo`) co obiekty, do których on się odwołuje (tabela `Address`), **serwer SQL sprawdza jedynie uprawnienia użytkownika do bezpośrednio wywołanego przez niego obiektu**. Rozwiązanie to pozwala odebrać wszystkim użytkownikom uprawnienia do tabel i udostępnić im je poprzez widoki, funkcje i procedury składowane.

Łańcuchy własności dotyczą jednak wyłącznie instrukcji `SELECT`, `INSERT`, `UPDATE` i `DELETE` wykonywanych w ramach jednej bazy danych. Dlatego próba wykonania przez użytkownika `Szelor` procedury obcinającej tabelę skończy się niepowodzeniem:

```
SELECT *
INTO Tab
FROM Person.Address;
GO
CREATE PROCEDURE Person.TrunTab
AS
TRUNCATE TABLE Tab;
GO
```

```
GRANT EXECUTE ON Person.TrunTab
TO Szelor;
GO
EXECUTE AS USER ='Szelor';
EXEC Person.TrunTab;
---------------------------------------
Msg 1088, Level 16, State 7, Procedure TrunTab, Line 3
Cannot find the object "Tab" because it does not exist or you do not have permissions.
```

Jeżeli jednak procedura będzie wykonywana w kontekście użytkownika `Admin`, a nie użytkownika, który ją wywołał, a użytkownik `Szelor` będzie miał prawo do posługiwania się tożsamością użytkownika `dbo`, zostanie ona wykonana poprawnie:

```
REVERT;
CREATE LOGIN Admin
WITH PASSWORD='Raz2Trzy!';
GO
CREATE USER Admin
FOR LOGIN Admin;
GO
EXEC sp_addrolemember 'db_owner', 'Admin';
GO
ALTER PROCEDURE Person.TrunTab
WITH EXECUTE AS 'Admin'
AS
TRUNCATE TABLE Tab;
GO
GRANT EXECUTE ON Person.TrunTab
TO Szelor;
GRANT IMPERSONATE ON USER::dbo
TO Szelor;
GO
EXECUTE AS USER ='Szelor';
EXEC Person.TrunTab;
---------------------------------------
Command(s) completed successfully.
```

Kryptografia

Serwer SQL 2008 umożliwia szyfrowanie oraz cyfrowe podpisywanie danych i modułów kodu, jednocześnie zarządza kluczami w taki sposób, że operacje kryptograficzne mogą być wykonywane automatycznie, bez wiedzy użytkowników. **Edycja Enterprise dodatkowo pozwala na zaszyfrowanie całych baz danych w sposób niewidoczny nie tylko dla ich użytkowników, ale również niewymagający żadnych zmian aplikacji klienckich**.

Dostawcy usług kryptograficznych

Serwer SQL 2008 korzysta z algorytmów kryptograficznych udostępnianych przez system operacyjny, w którym działa[8]. Ponieważ w systemach Windows dostawcy usług kryptograficznych mogą być dodawani lub usuwani przez administratora, lista dostępnych algorytmów zależy od wersji systemu operacyjnego i jego konfiguracji.

Algorytmy symetryczne

Algorytmy tego typu albo używają tego samego klucza do szyfrowania i deszyfrowania wiadomości, albo klucz deszyfrujący można bezpośrednio wyprowadzić z klucza szyfrującego. Główną zaletą algorytmów symetrycznych jest ich szybkość — są one kilka tysięcy razy szybsze od algorytmów asymetrycznych.

Natomiast ich podstawową wadą jest konieczność udostępnienia wszystkim użytkownikom systemu odpowiednich kluczy. Dlatego **algorytmy symetryczne powinny być używane do szyfrowania danych, a ich klucze powinny być szyfrowane algorytmami asymetrycznymi**. W ten sposób połączymy zalety obu typów algorytmów — kluczami asymetrycznymi zaszyfrujemy niewielkie klucze symetryczne, a te z kolei będą używane do wydajnego szyfrowania dużych ilości danych.

Najbezpieczniejszym i zalecanym algorytmem symetrycznym jest AES[9] (ang. *Advanced Encryption Standard*). Serwer SQL 2008 pozwala na stosowanie kluczy o długościach 256, 196 lub 128 bitów. Wydajność algorytmu AES zmniejsza się wraz ze wzrostem długości klucza i dla kluczy 256- jest około 20% niższa niż dla kluczy 128-bitowych.

Nie należy używać algorytmu RC4. RC4 jest szyfrem strumieniowym, w którym bezpieczeństwo szyfrogramów zależy od właściwego zarządzania kluczami, szczególnie na niedopuszczeniu do ponownego użycia tego samego klucza[10]. W przeciwnym razie atakujący, wykonując operację `XOR` na dwóch różnych szyfrogramach, które zostały zaszyfrowane tym samym kluczem, uzyska informacje umożliwiające mu złamanie szyfrogramów, a nawet poznanie samego klucza. **Implementacja RC4 w serwerze SQL 2008 wielokrotnie używa tego samego klucza.**

[8] Tylko edycja Enterprise umożliwia rejestrowanie dodatkowych dostawców usług kryptograficznych, co zostało przedstawione w dalszej części rozdziału.

[9] Algorytm AES jest niedostępny w systemach Windows XP. W ich przypadku pozostaje nam stosowanie algorytmu 3DES.

[10] W przypadku algorytmów blokowych, które działają w jakimś trybie szyfru blokowego (w serwerze SQL 2005 używany jest tryb łańcuchowego szyfru blokowego CBC, w którym każdy blok tekstu jawnego przed zaszyfrowaniem jest przekształcany funkcją `XOR` z szyfrogramem uzyskanym poprzez zaszyfrowanie poprzedniego bloku wiadomości), niewłaściwe zarządzanie kluczami jest mniej niebezpieczne.

Algorytmy asymetryczne

Jedyny dostępny w serwerze SQL 2008 algorytm asymetryczny (czyli taki, w którym do szyfrowania danych używa się innego klucza niż do ich odszyfrowania), RSA, został opracowany przez Rivesta, Shamira i Adlemana w 1968 — jest to pierwszy i do dziś najpopularniejszy algorytm asymetryczny. W przeciwieństwie do szyfrów asymetrycznych, których działanie polega na powtarzaniu dwóch podstawowych w szyfrowaniu operacji mieszania i przestawiania bitów, algorytm RSA wykorzystuje faktoryzację dużych liczb pierwszych, czyli funkcję, dla której nie znamy efektywnej metody odwrócenia kierunku obliczeń.

Serwer SQL 2008 pozwala na stosowanie kluczy o długościach 512, 1024 i 2048 bitów. Ponieważ szyfrogram RSA 512 można złamać na domowym komputerze w ciągu kilku godzin, a szyfrogram 1024-bitowy prawdopodobnie zostanie złamany w ciągu kilku lat, należy używać kluczy o długości 2048 bitów.

Wielkość liczb i natura obliczeń powodują, że algorytm RSA jest tysiące razy wolniejszy od dowolnego algorytmu symetrycznego. Oznacza to, że jeżeli zaszyfrowanie pewnych danych algorytmem AES trwało kilka sekund, to zaszyfrowanie tych samych danych algorytmem RSA zajęłoby kilka godzin.

Algorytmy mieszania

Funkcje mieszania (ang. *hash*) są funkcjami jednokierunkowymi, zwracającymi wyniki o tej samej, niezależnej od rozmiaru danych wejściowych, długości. Jedynym sposobem na odtworzenie oryginalnych danych na podstawie znanego wyniku funkcji mieszania jest wyliczenie jej dla wszystkich możliwych danych. Jeżeli uzyskamy taki sam wynik, oryginalne dane prawie na pewno też będą takie same[11].

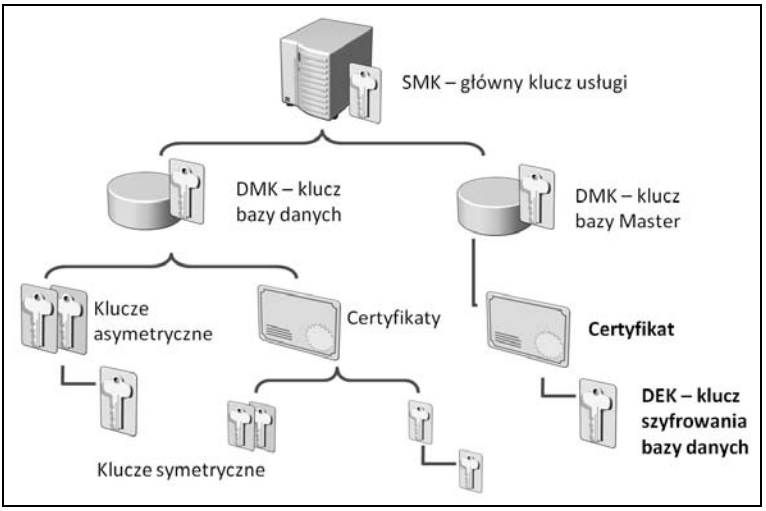
Funkcje mieszania nie umożliwiają szyfrowania danych, ale doskonale nadają się do sprawdzania, czy nie zostały one zmodyfikowane. W tym celu wystarczy zapisać sygnaturę oryginalnych danych, a następnie podczas ich odczytywania raz jeszcze przeprowadzić te same obliczenia i porównać obie sygnatury. Jeżeli będą identyczne, to znaczy, że dane nie zostały zmodyfikowane.

Niestety, żadna z używanych obecnie funkcji mieszania nie gwarantuje bezpieczeństwa. Względnie najbezpieczniejszą z dostępnych w serwerze SQL 2008 funkcji jest SHA-1.

[11] Kryptograficzne funkcje mieszania powinny cechować: maksymalna losowość wyniku, minimalna szansa zwrócenia tego samego wyniku dla różnych wiadomości i maksymalne zróżnicowanie wyniku dla niewielkich zmian wiadomości.

Hierarchia kluczy

Serwer SQL 2008 umożliwia automatyczne zarządzanie kluczami w taki sposób, że użytkownicy mogą szyfrować i deszyfrować dane bez podawania dodatkowych kluczy czy chroniących je haseł. Jednocześnie poziom bezpieczeństwa szyfrogramów pozostaje niezmieniony, tj. tylko administratorzy systemu i serwera SQL mogą je odszyfrować (rysunek 7.3).



Rysunek 7.3. *Hierarchia kluczy kryptograficznych z zaznaczonymi pogrubioną czcionką kluczami szyfrowania baz danych DEK (ang. Database Encryption Key)*

Główny klucz usługi

Podczas instalowania serwera SQL 2008 generowany jest główny klucz usługi SMK (ang. *Service Master Key*), który po zaszyfrowaniu algorytmem 3DES zapisywany jest w bazie master. Klucz SMK jest zaszyfrowany algorytmem 3DES i chroniony przez DPAPI (ang. *Data Protection Application Programming Interface*) systemu Windows.

Klucz pozwalający odszyfrować klucz SMK jest chroniony przez podsystem Windows DPAPI (ang. *Data Protection API*): jedna jego kopia zapisywana jest w magazynie komputera, a druga — w magazynie użytkownika, z uprawnieniami którego działa usługa serwera SQL. Dzięki czemu klucz SMK może być automatycznie odszyfrowany, ale tylko w środowisku oryginalnego systemu operacyjnego — jeżeli np. ktoś spróbuje podłączyć dysk z zainstalowanym serwerem SQL 2008 do innego komputera, odszyfrowanie klucza SMK będzie niemożliwe. Podczas każdego uruchomienia serwera SQL 2008 sprawdzana jest poprawność obu zaszyfrowanych kopii klucza i jeżeli jedna z nich okaże się niedostępna, zostanie odtworzona na podstawie drugiej, poprawnej kopi.

Klucz SMK jest bezpośrednio lub pośrednio używany do szyfrowania i odszyfrowywania wszystkich innych kluczy kryptograficznych. Chroni on główne klucze bazy danych DMK (ang. *Database Master Key*) oraz poufne dane serwera, takie jak dodatkowe poświadczenia oraz hasła loginów do powiązanych serwerów.

Klucz SMK może być wyeksportowany, a plik jego kopii powinien być przechowywany w bezpiecznym miejscu:

```
BACKUP SERVICE MASTER KEY TO FILE = 'h:\SMK.KEY'
ENCRYPTION BY PASSWORD = 'T0Hasl,.oMusi%$3BykjcNapraw890$#$deBe=-"zpiecz89nE!';
```

Możliwa jest również zmiana klucza SMK. Operacja ta przebiega w trzech etapach i jeżeli dowolny etap zakończy się błędem, cała procedura zostanie przerwana, a klucz SMK — niezmieniony:

1. Najpierw wygenerowany zostaje nowy klucz SMK.
2. Następnie odszyfrowane zostają wszystkie dane zaszyfrowane poprzednim kluczem SMK.
3. Na końcu te dane zostają zaszyfrowane nowym kluczem SMK:

```
ALTER SERVICE MASTER KEY REGENERATE;
```

Kopię klucza SMK można odtworzyć, również na innym serwerze SQL 2008, wykonując instrukcję `RESTORE`:

```
RESTORE SERVICE MASTER KEY FROM FILE = 'h:\SMK.KEY'
DECRYPTION BY PASSWORD = 'T0Hasl,.oMusi%$3BykjcNapraw890$#$deBe=-"zpiecz89nE!';
```

Odtworzenie klucza przypomina jego zmianę, jednak serwer SQL 2008 nie wygeneruje nowego klucza SMK, ale odczyta go z pliku kopii. Następnie odszyfrowane zostaną wszystkie dane chronione starym kluczem SMK i ponownie zaszyfrowane odzyskanym kluczem.

W ostateczności możemy wymusić zmianę klucza SMK, nawet jeżeli niektóre zaszyfrowane nim dane zostaną w wyniku tej zmiany nieodwracalnie utracone. Opcja `FORCE` może być użyta zarówno przy zmianie, jak i odtworzeniu klucza SMK:

```
ALTER SERVICE MASTER KEY
FORCE REGENERATE;
-- lub
RESTORE SERVICE MASTER KEY
FROM FILE = 'h:\SMK.KEY'
DECRYPTION BY PASSWORD = 'T0Hasl,.oMusi%$3BykjcNapraw890$#$deBe=-"zpiecz89nE!'
FORCE;
```

Główny klucz bazy danych

W każdej bazie danych administrator może utworzyć klucz DMK. Domyślnie jedna kopia tego klucza jest szyfrowana kluczem SMK i zapisywana w bazie master, a druga — podanym przy jego tworzeniu hasłem i zapisywana w bazie, w której klucz został utworzony. Obie kopie klucza DMK szyfrowane są algorytmem 3DES:

```
CREATE DATABASE r07;
GO
USE r07;
GO
CREATE MASTER KEY
ENCRYPTION BY PASSWORD = 'ToPowinnoBycInnE,ROwnieSilneH@sl0!';
```

```
SELECT name, algorithm_desc
FROM sys.symmetric_keys;
--------------------------------------
##MS_DatabaseMasterKey##       TRIPLE_DES
```

Kluczem DMK szyfrowane są wystawiane w tej bazie danych klucze asymetryczne oraz certyfikaty. Tak więc do odszyfrowania certyfikatu potrzebny jest klucz DMK, do odszyfrowania którego potrzebny jest z kolei klucz SMK. Warto więc wyeksportować klucz DMK do pliku i przechowywać go w bezpiecznym miejscu:

```
BACKUP MASTER KEY TO FILE = 'C:\TEMP\DBK.KEY'
ENCRYPTION BY PASSWORD = 'KO9jRm,YsdeSiopEIOOM+_IdfsSI43E';
```

Nie można usunąć kopii klucza DMK chronionej podanym przy jego tworzeniu hasłem. Można natomiast skasować kopię klucza DMK chronioną kluczem SMK. Uniemożliwi to automatyczny dostęp do klucza DMK i wszystkich zaszyfrowanych nim kluczy. **Jeżeli administrator serwera SQL nie będzie znał tego hasła, uniemożliwimy mu posługiwanie się kluczami użytkowników**:

```
CREATE CERTIFICATE Cert1
WITH SUBJECT = 'Test odszyfrowania DBK przez SMK';
SELECT name, pvt_key_encryption_type_desc
FROM sys.certificates;
---------------------------------------
Cert1    ENCRYPTED_BY_MASTER_KEY

ALTER MASTER KEY
DROP ENCRYPTION BY SERVICE MASTER KEY;
CREATE CERTIFICATE Cert2
WITH SUBJECT = 'Test odszyfrowania DBK przez SMK';
---------------------------------------
Msg 15581, Level 16, State 3, Line 3
Please create a master key in the database or open the master key in the session
↪before performing this operation.
```

Od teraz użycie klucza DBK wymaga jego otwarcia przy użyciu hasła:

```
OPEN MASTER KEY DECRYPTION BY PASSWORD = 'ToPowinnoBycInnE,ROwnieSilneH@sl0!';
CREATE CERTIFICATE Cert2
WITH SUBJECT = 'Test odszyfrowania DBK przez SMK';
CLOSE MASTER KEY;
---------------------------------------
Command(s) completed successfully.
```

Ponieważ każda instancja serwera SQL 2008 ma niepowtarzalny klucz SMK, przeniesienie bazy z jednego serwera na inny spowoduje, że klucz DBK tej bazy nie będzie mógł być automatycznie odszyfrowany. Żeby przywrócić automatyczne zarządzanie kluczami, należy samodzielnie odtworzyć klucz DBK przeniesionej bazy danych i utworzyć jego kopię zaszyfrowaną kluczem SMK nowego serwera:

```
OPEN MASTER KEY DECRYPTION BY PASSWORD = 'ToPowinnoBycInnE,ROwnieSilneH@sl0!';
ALTER MASTER KEY
ADD ENCRYPTION BY SERVICE MASTER KEY;
CLOSE MASTER KEY;
```

Certyfikaty i klucze asymetryczne użytkowników

Następne w hierarchii kluczy kryptograficznych są klucze RSA i certyfikaty użytkowników. Serwer SQL 2008 używa certyfikatów jedynie jako pojemników na klucze asymetryczne — generowany przez niego klucz prywatny certyfikatu ma zawsze długość 1024 bitów, a długość importowanego klucza prywatnego musi być większa niż 384 i mniejsza niż 3456 bitów:

```
CREATE USER Szelor
FOR LOGIN Szelor;
CREATE CERTIFICATE Cert3
AUTHORIZATION Szelor
WITH SUBJECT = 'Certyfikat chroniący klucz symetryczny';

CREATE ASYMMETRIC KEY AKey1
WITH ALGORITHM = RSA_2048
ENCRYPTION BY PASSWORD = 'WtOreK11:43';
```

Klucze symetryczne użytkowników

Ostatnie w hierarchii kluczy kryptograficznych są klucze symetryczne używane do szyfrowania i deszyfrowania danych. Chroniąc klucze symetryczne certyfikatami, które z kolei są chronione kluczem DBK, uzyskamy łatwy w zarządzaniu i jednocześnie wydajny, hybrydowy system kryptograficzny:

```
CREATE SYMMETRIC KEY SKey1
AUTHORIZATION Szelor
WITH ALGORITHM = AES_128
ENCRYPTION BY CERTIFICATE Cert3;
```

Przenoszenie kluczy użytkowników

Decydując się na używanie funkcji kryptograficznych serwera SQL 2008, powinniśmy też zaplanować archiwizację i przenoszenie pomiędzy bazami kluczy użytkowników. Umożliwi to m.in. korzystanie z testowych baz danych.

> Klucze użytkowników, tak jak klucz DMK, zapisywane są w tabelach systemowych bazy danych, w których zostały utworzone. Kopie zapasowe bazy będą więc zawierały wszystkie klucze jej użytkowników. W celu umożliwienia użytkownikom normalnej pracy z odtworzoną na innym serwerze kopią bazy wystarczy odtworzenie na nim głównego klucza usługi serwera SQL.

Eksport i import certyfikatów

Certyfikaty oraz klucze prywatne można wyeksportować do plików:

```
BACKUP CERTIFICATE Cert3
TO FILE = 'C:\TEMP\Cert3.cer'
WITH PRIVATE KEY
(FILE = 'C:\TEMP\SKey1.KEY',
ENCRYPTION BY PASSWORD = 'DE5@#$vfr^dEy*(&DftS)');
```

Ich odtworzenie w innej bazie danych wymaga więc jedynie wcześniejszego utworzenia w niej klucza DMK:

```
USE tempdb
CREATE MASTER KEY
ENCRYPTION BY PASSWORD = 'KluczB@zyTemp';
CREATE CERTIFICATE Cert
FROM FILE = 'C:\TEMP\Cert3.cer'
WITH PRIVATE KEY
(FILE = 'C:\TEMP\SKey1.KEY',
DECRYPTION BY PASSWORD = 'DE5@#$vfr^dEy*(&DftS)')
```

Tworzenie duplikatów kluczy symetrycznych

Kluczy symetrycznych nie można wyeksportować do plików. Jeżeli chcemy odtworzyć klucz symetryczny w innej bazie danych, będziemy musieli utworzyć jego wierną kopię. W tym celu należy podczas tworzenia duplikatu klucza podać takie same wartości parametrów `ALGORITHM`, `KEY_SOURCE` i `IDENTITY_VALUE`:

1. Tworząc klucz, określa się algorytm i jego długość. W ten sposób wybieramy używany do szyfrowania i deszyfrowania algorytm, a więc jeżeli chcemy odtworzyć ten sam klucz kryptograficzny, musi on korzystać z tego samego algorytmu.
2. Parametr `KEY_SOURCE` określa źródło klucza. Jeżeli go nie podamy, serwer SQL do wygenerowania klucza wykorzysta pseudolosowe dane. **Ponieważ na podstawie tych samych danych źródłowych zawsze generowany jest ten sam klucz, muszą one być chronione równie dobrze jak sam klucz symetryczny**.
3. Serwer SQL posługuje się identyfikatorami kluczy, a nie ich opisowymi nazwami. Identyfikator klucza też zostanie wygenerowany na podstawie pseudolosowych danych, chyba że użyty zostanie parametr `IDENTITY_VALUE`. Wtedy identyfikator klucza zostanie wyliczony na podstawie wartości tego parametru. Ponieważ identyfikator klucza nie musi być chroniony, użyte do jego wygenerowania dane również nie muszą być poufne.

Sposób wykonania kopii klucza symetrycznego przedstawia poniższy przykład:

1. W oryginalnej bazie wystawiamy klucz symetryczny chroniony certyfikatem i sprawdzamy jego identyfikator:

```
USE r07
CREATE SYMMETRIC KEY SKey2
WITH ALGORITHM = AES_128,
KEY_SOURCE = 'KDF9384939834kjf02930sd9f09r04395,c03945039102,
↪c59mncxvSZUIE49CVGKSL38493KALSA;W=1-2',
IDENTITY_VALUE = 'Klucz SKey2'
ENCRYPTION BY CERTIFICATE Cert3;

SELECT key_guid
FROM sys.symmetric_keys
WHERE name = 'Skey2';
------------------------------------
5DD0C400-56A3-0DB4-C043-2A1F4E8E8A80
```

2. Tego klucza używamy do zaszyfrowania przykładowych danych:

```
CREATE TABLE Tab1
(Kol1 varbinary(100));
GO
OPEN SYMMETRIC KEY SKey2
DECRYPTION BY CERTIFICATE Cert3;
INSERT INTO Tab1
VALUES (EncryptByKey(Key_GUID('SKey2'),'Tajne dane'));
CLOSE ALL SYMMETRIC KEYS;
```

3. Następnie przenosimy zaszyfrowane dane do innej bazy:

```
SELECT *
INTO tempdb..Tab1
FROM Tab1
```

4. Żeby można było odczytać skopiowane szyfrogramy, musimy w pierwszej kolejności wyeksportować certyfikat i klucz prywatny chroniące klucz symetryczny i odtworzyć je w docelowej bazie:

```
BACKUP CERTIFICATE Cert3
TO FILE = 'c:\Temp\Cert3.cer'
WITH PRIVATE KEY
(FILE = 'c:\Temp\KeyCert3.KEY',
ENCRYPTION BY PASSWORD = 'DE5@(*DvfSr^dE');

USE tempdb
CREATE MASTER KEY
ENCRYPTION BY PASSWORD = 'Structured Query Language!';
CREATE CERTIFICATE Cert3
FROM FILE = 'c:\Temp\Cert3.cer'
WITH PRIVATE KEY
(FILE = 'c:\Temp\KeyCert3.KEY',
DECRYPTION BY PASSWORD = 'DE5@(*DvfSr^dE');
```

5. Następnie tworzymy duplikat klucza symetrycznego:

```
CREATE SYMMETRIC KEY SKey2
WITH ALGORITHM = AES_128,
KEY_SOURCE = 'KDF9384939834kjf02930sd9f09r04395,c03945039102,
➥c59mncxvSZUIE49CVGKSL38493KALSA;W=1-2',
IDENTITY_VALUE = 'Klucz SKey2'
ENCRYPTION BY CERTIFICATE Cert3;
```

6. Ponieważ klucz i jego identyfikator są identyczne, użytkownik będzie mógł odczytać skopiowany szyfrogram:

```
OPEN SYMMETRIC KEY SKey2
DECRYPTION BY CERTIFICATE Cert3;
SELECT CAST(DecryptByKey(Kol1) AS VARCHAR(50))
FROM dbo.Tab1;
CLOSE ALL SYMMETRIC KEYS;
---------------------------------------
Tajne dane
```

Elastyczne zarządzanie kluczami (dotyczy edycji Enterprise)

Serwer SQL 2008 Enterprise Edition umożliwia przechowywanie kluczy kryptograficznych na zewnętrznych urządzeniach, takich jak napędy USB lub karty inteligentne. Pozwala to fizycznie oddzielić szyfrogramy od umożliwiających ich odszyfrowanie kluczy.

Edycja Enterprise może też korzystać z kluczy wygenerowanych przez zewnętrzne systemy kryptograficzne. Dzięki czemu zintegrujemy szyfrowanie baz danych z istniejącymi już w firmie rozwiązaniami, zautomatyzujemy dystrybucję kluczy (w tym ich przedawnianie) oraz będziemy mogli używać sprzętowych modułów szyfrujących.

Skorzystanie z tych funkcjonalności wymaga ustawienia jednej zaawansowanej opcji serwera i zarejestrowania nowego dostawcy usług kryptograficznych:

```
EXEC sp_configure 'show advanced', 1;
RECONFIGURE;
EXEC sp_configure 'EKM provider enabled', 1;
RECONFIGURE;

CREATE CRYPTOGRAPHIC PROVIDER nazwa
FROM FILE = <plik.dll>;
```

Szyfrowanie danych

Zaszyfrowane dane przechowywane są w postaci binarnej. Ich rozmiar w bajtach można obliczyć z dokładnością do jednego bloku ze wzoru: *((8 + dane) / (blok+1)) * (2 * blok + 16)*. Rozmiar bloku wynosi 8 bajtów dla algorytmu 3DES i 16 bajtów w przypadku algorytmu AES. **W praktyce należy dodać do wielkości szyfrowanych danych 48 bajtów dla algorytmu 3DES i 66 bajtów dla algorytmu AES**.

Do odszyfrowania lub zaszyfrowania danych użytkownik musi mieć nadane uprawnienia do odczytywania lub modyfikowania danych oraz prawo posługiwania się odpowiednim kluczem. W ten sposób szyfrowanie danych jest elementem systemu zabezpieczeń na poziomie wierszy (ang. *Row Level Security*).

Utworzone poprzednio klucze: symetryczny, chroniący go certyfikat oraz klucze DMK i SMK zostaną przez nas użyte do zaszyfrowania skopiowanych do bazy r07 danych dotyczących zamówień:

1. Zaczniemy od skopiowania największej, liczącej ponad 120 MB tabeli bazy AdventureWorks2008 i dodania do niej kolumny, w której zapiszemy szyfrogramy:

```
USE r07
SELECT *
INTO dbo.SalesOrderDetailEncrypted
FROM AdventureWorks2008.Sales.SalesOrderDetail;
ALTER TABLE dbo.SalesOrderDetailEncrypted
ADD CarrierTrackingNumberEncrypted VARBINARY(91);
```

2. Następnie zaszyfrujemy numery spedycyjne zamówień. Ponieważ użyjemy do tego klucza symetrycznego, należy go otworzyć, posługując się odpowiednim certyfikatem, zaszyfrować dane systemową funkcją `EncryptByKey` i zamknąć klucz symetryczny:

```
OPEN SYMMETRIC KEY Skey1
DECRYPTION BY CERTIFICATE Cert3;
UPDATE dbo.SalesOrderDetailEncrypted
SET CarrierTrackingNumberEncrypted =
EncryptByKey(Key_GUID('SKey1'),CarrierTrackingNumber);
CLOSE SYMMETRIC KEY Skey1;
```

3. Możemy już usunąć kolumnę z jawną postacią numerów spedycyjnych:

```
ALTER TABLE dbo.SalesOrderDetailEncrypted
DROP COLUMN CarrierTrackingNumber;
```

4. Od teraz odczytanie numerów spedycyjnych wymaga ich odszyfrowania, a więc otwarcia klucza symetrycznego (tylko właściciel i osoba, której zostały nadane takie uprawnienia, mogą posługiwać się kluczem kryptograficznym), wywołania funkcji `DecryptByKey` i zamknięcia klucza:

> Otwarcie klucza wymaga jego odszyfrowania za pomocą hasła albo, jak w tym przykładzie z wykorzystaniem hierarchii kluczy, serwera SQL.

```
OPEN SYMMETRIC KEY Skey1
DECRYPTION BY CERTIFICATE Cert3;
SELECT CAST(DecryptByKey(CarrierTrackingNumberEncrypted) AS NVARCHAR(25)),
➥CarrierTrackingNumberEncrypted, LEN(CarrierTrackingNumberEncrypted)
FROM dbo.SalesOrderDetailEncrypted
WHERE SalesOrderID = 50256;
CLOSE ALL SYMMETRIC KEYS;
----------------------------------------
FE9D-4B76-BB      0x00EF2EF7AA70C74AB357BC8669C0F81201000000B718D9FD6…     84
```

Wydajność

Te same dane kilkukrotnie zaszyfrowane tym samym kluczem dadzą różne szyfrogramy[12]. Żeby się o tym przekonać, wystarczy wykonać poniższy skrypt (zwracająca sygnaturę funkcja `CHECKSUM` została użyta tylko dla skrócenia i poprawy czytelności wyniku):

```
USE tempdb
CREATE SYMMETRIC KEY Test
WITH ALGORITHM = AES_128
ENCRYPTION BY PASSWORD = 'L(*U%^b*&8I)(*E...';
OPEN SYMMETRIC KEY Test
DECRYPTION BY PASSWORD = 'L(*U%^b*&8I)(*E...';
PRINT CHECKSUM(EncryptByKey(Key_GUID('Test'),'Dawno temu...'));
PRINT CHECKSUM(EncryptByKey(Key_GUID('Test'),'Dawno temu...'));
```

[12] Wyjątkiem jest algorytm RC4.

```
PRINT CHECKSUM(EncryptByKey(Key_GUID('Test'),'Dawno temu...'));
PRINT CHECKSUM(EncryptByKey(Key_GUID('Test'),'Dawno temu...'));
CLOSE SYMMETRIC KEY Test;
---------------------------------------
-919475613
-1888524620
-1431208946
1606422219
```

Poprawia to bezpieczeństwo szyfrogramów i uniemożliwia ich porównywanie[13]. Niestety, powoduje także, że przeszukiwanie szyfrogramów jest niemożliwe, a więc serwer SQL 2008 będzie musiał za każdym razem odczytać wszystkie szyfrogramy, odszyfrować je i przeszukać tak otrzymane wyniki. W praktyce oznacza to, że indeksy założone na zaszyfrowanych kolumnach i tak nie będą używane, co pokazuje kolejny przykład:

1. Poindeksujemy istniejące wartości szyfrogramów. Ponieważ są one uzupełniane o pseudolosowe ziarno (ang. *Seed*) możemy założyć niepowtarzalność kluczy indeksu:

```
USE r07
CREATE UNIQUE INDEX I_Encrypted
ON dbo.SalesOrderDetailEncrypted(CarrierTrackingNumberEncrypted)
WHERE CarrierTrackingNumberEncrypted IS NOT NULL;
```

2. Jeżeli odczytamy wszystkie szyfrogramy, serwer SQL 2008 przeskanuje indeks, ale tego typu zapytania nie są w praktyce używane:

```
SET SHOWPLAN_TEXT ON
GO
SELECT CarrierTrackingNumberEncrypted
FROM dbo.SalesOrderDetailEncrypted
WHERE CarrierTrackingNumberEncrypted IS NOT NULL;
---------------------------------------
 |--Index Scan(OBJECT:([r07].[dbo].[SalesOrderDetailEncrypted].[I_Encrypted]))
```

3. Natomiast próba przeszukania szyfrogramów realizowana jest, co prawda, w oparciu o indeks, ale zawsze kończy się nieznalezieniem pasujących wierszy. Z powodu zmiennego ziarna zapytanie, które powinno zwrócić dane istniejącego zamówienia, zwróciło zero wierszy:

```
SET SHOWPLAN_TEXT OFF
GO
OPEN SYMMETRIC KEY Skey1
DECRYPTION BY CERTIFICATE Cert3;
SELECT SalesOrderID, CAST(DecryptByKey(CarrierTrackingNumberEncrypted)
↪AS NVARCHAR(25))AS Szyfrogram
FROM dbo.SalesOrderDetailEncrypted
WHERE CarrierTrackingNumberEncrypted = EncryptByKey(Key_GUID('SKey1'),
↪'FE9D-4B76-BB');
CLOSE ALL SYMMETRIC KEYS;
---------------------------------------
NULL
```

[13] W przeciwnym przypadku atakujący mógłby przez proste porównanie szyfrogramów dowiedzieć się np., którzy klienci mają taki sam rabat.

4. Jedyne rozwiązanie polega na odszyfrowaniu i porównaniu jawnej postaci danych, ale to uniemożliwia przeszukanie indeksu[14]:

```
SET SHOWPLAN_TEXT ON
GO
OPEN SYMMETRIC KEY Skey1
DECRYPTION BY CERTIFICATE Cert3;
SELECT SalesOrderID, CAST(DecryptByKey(CarrierTrackingNumberEncrypted)
↪AS NVARCHAR(25))
FROM dbo.SalesOrderDetailEncrypted
WHERE CAST(DecryptByKey(CarrierTrackingNumberEncrypted) AS NVARCHAR(25))
↪= 'FE9D-4B76-BB';
CLOSE ALL SYMMETRIC KEYS;
----------------------------------------
 |--Filter(WHERE:([Expr1004]=N'FE9D-4B76-BB'))
 |--Compute Scalar(DEFINE:([Expr1004]=CONVERT(nvarchar(25),DecryptByKey([r07].
 ↪[dbo].[SalesOrderDetailEncrypted].[CarrierTrackingNumberEncrypted]),0)))
 |--Table Scan(OBJECT:([r07].[dbo].[SalesOrderDetailEncrypted]))
```

Sprawdzanie autentyczności

Potwierdzanie autentyczności danych jest często ważniejsze niż zapewnianie im poufności. Przykładowo informacje o zamówieniach mogą być bez większej szkody ujawnione niepowołanym osobom, ale możliwość ich modyfikowania przez te osoby mogłaby spowodować ogromne straty.

Jeszcze groźniejsza byłaby zmiana kodu działającej z uprawnieniami administracyjnymi procedury — dlatego coraz więcej producentów podpisuje cyfrowo swoje programy. Serwer SQL 2008 pozwala w podobny sposób podpisać procedury i funkcje bazodanowe.

Weryfikowanie autentyczności danych

Tak jak szyfrowanie jest powszechnie stosowaną techniką zapewniania poufności danych, tak podpisy cyfrowe są standardowym sposobem sprawdzania ich autentyczności. W najprostszym przypadku można by wyliczyć sygnatury niezaszyfrowanych danych i zapisać je w dodatkowej kolumnie tabeli. Podczas odczytywania danych wystarczyłoby odczytać ich sygnatury i porównać je z sygnaturami wyliczonymi na podstawie aktualnych danych. Gdyby sygnatury się różniły, znaczyłoby to, że dane zostały w międzyczasie zmodyfikowane.

[14] W tym przypadku tylko jeden wiersz pasuje do warunku wyszukiwania, a mimo to serwer SQL przeskanował całą tabelę, odszyfrował wszystkie numery spedycyjne i sprawdził, który z nich odpowiada szukanemu.

Wyliczenie sygnatur danych ma jednak dwa słabe punkty:

1. Przede wszystkim atakujący mógłby samodzielnie uaktualnić sygnatury i w ten sposób ukryć modyfikację danych[15].
2. Zaszyfrowanie podpisanych danych osłabia siłę szyfrogramów — atakujący mógłby wygenerować wszystkie możliwe dane, wyliczyć ich sygnatury i porównać je z sygnaturami zapisanymi w tabeli.

Rozwiązanie obu problemów polega na dodaniu do oryginalnych danych dowolnego nieznanego atakującemu sekretu i wyliczeniu ich sygnatur po tej zmianie. Sygnatura wyliczona na podstawie oryginalnych danych uzupełnionych o wektor inicjujący IV (ang. *Initialization Vector*) nazywana jest kodem autentyczności danych MAC (ang. *Message Authentication Code*).

Przykładowa implementacja sprawdzania kodów MAC została pokazana poniżej:

1. Używane do podpisywania danych wektory inicjujące będą po zaszyfrowaniu przechowywane w osobnej tabeli. Żeby utrudnić atak słownikowy[16], każda tabela będzie miała inny wektor inicjujący:

```
CREATE TABLE tIV
(TblID INT PRIMARY KEY,
IV VARBINARY(100) NOT NULL);
```

2. Do szyfrowania użyjemy klucza AES i certyfikatu:

```
CREATE CERTIFICATE CertIV
WITH SUBJECT = 'Ochrona wektorow IV';
CREATE SYMMETRIC KEY SKeyIV
WITH ALGORITHM = TRIPLE_DES
ENCRYPTION BY CERTIFICATE CertIV;
```

3. Pseudolosowy, zwrócony przez funkcję `NEWID` wektor IV zaszyfrujemy i zapiszemy w przygotowanej tabeli:

```
OPEN SYMMETRIC KEY SKeyIV
DECRYPTION BY CERTIFICATE CertIV
INSERT INTO tIV
VALUES (1,EncryptByKey(Key_GUID('SKeyIV'),CAST(NEWID() AS VARCHAR(100))))
CLOSE ALL SYMMETRIC KEYS
```

4. Atakujący, nie mając dostępu do klucza `SKeyIV`, będzie mógł co najwyżej odczytać zaszyfrowany wektor inicjujący:

```
SELECT *
FROM tIV
----------------------------------------
1  0x00370CCD6E41AB4EA104B6E90713DAD301000000AB603258B9F7D37D432729CB9F330D39DF
↪10D74B0B601DBA9F79DF1B8710A701068A65181A6723AED33E7DF4146756F097FAAF40368B73D7
```

[15] Tworząc bezpieczny system, należy założyć, że każdy jego element może być złamany. Dlatego działanie podsystemu odpowiedzialnego za sprawdzanie autentyczności danych nie powinno zależeć od uprawnień do tabel nadanych użytkownikom.

[16] Popularna technika odwrócenia funkcji skrótu polega na wykorzystaniu tęczowej tabeli (ang. *Rainbow Table*) zawierającej wszystkie możliwe wartości tej funkcji dla danego ziarna.

5. Natomiast osoby upoważnione do generowania sygnatur użyją funkcji, która do podpisywanych danych dołączy odszyfrowany wektor IV, a następnie wyliczy ich sygnaturę[17]:

```
CREATE FUNCTION udfMac(@Dane NVARCHAR(4000), @TblID INT )
RETURNS BINARY(24) AS
BEGIN
 DECLARE @Key VARBINARY(100) = NULL
 SELECT @Key = DecryptByKeyAutoCert(cert_id('CertIV'), NULL, IV)
 FROM tIV
 WHERE TblID = @TblID
 RETURN (HashBytes( N'SHA1', CONVERT(VARBINARY(8000), @Dane) + @Key))
END
```

6. Po dodaniu do tabeli kolumny, w której zapisane zostaną sygnatury, możemy podpisać cyfrowo dane:

```
ALTER TABLE dbo.SalesOrderDetailEncrypted
ADD Mac BINARY(24);

UPDATE dbo.SalesOrderDetailEncrypted
SET Mac = dbo.udfMac (LineTotal,1);
```

7. Od teraz, porównując zapisane w tabeli i wyliczone na podstawie aktualnych danych sygnatury, znajdziemy zmodyfikowane dane (wykonując to ćwiczenie, przekonamy się, jak kosztowne i długotrwałe jest szyfrowanie i podpisywanie cyfrowe danych, i będziemy mogli porównać jego wydajność z wydajnością opisanego w dalszej części rozdziału szyfrowania baz danych)[18]:

```
UPDATE dbo.SalesOrderDetailEncrypted
SET LineTotal = 0.99
WHERE SalesOrderID = 53593;

SELECT SalesOrderID, LineTotal
FROM dbo.SalesOrderDetailEncrypted
WHERE Mac <> dbo.udfMac (LineTotal,1);
----------------------------------------
```

Weryfikowanie autentyczności modułów kodu

Podpisywanie modułów kodu to jedna z najpotężniejszych funkcji serwera SQL 2008. **Pozwala ona na czas wykonywania modułu nadać mu dodatkowe uprawnienia, bez konieczności zmieniania kontekstu jego wykonania**. Podpisując moduł kodu (np. procedurę), uzyskamy ten sam efekt, który uzyskalibyśmy podczas zmiany kontekstu jej wykonania za pomocą opcji `EXECUTE AS`, jednocześnie zachowując pełną kontrolę nad użytkownikami i możliwość monitorowania ich działań.

Jeżeli wymagane są dodatkowe uprawnienia na poziomie serwera, należy utworzyć certyfikat w bazie master, na jego podstawie wygenerować login i nadać temu loginowi wymagane uprawnienia, a następnie podpisać moduł certyfikatem.

[17] Żeby inni użytkownicy, nie tylko administrator, mogli używać tej funkcji, należy ją cyfrowo podpisać. Podpisywanie modułów kodu opisane zostało w następnym punkcie.

[18] Na testowym serwerze sprawdzenie autentyczności danych trwało ponad 15 minut.

Jeżeli natomiast chcemy nadać dodatkowe uprawnienia na poziomie bazy danych, należy w tej bazie utworzyć certyfikat, na jego podstawie wykreować konto użytkownika, nadać mu wymagane uprawnienia i podpisać moduł certyfikatem.

W obu przypadkach należy nadać odpowiednim użytkownikom prawo wykonywania tak podpisanego modułu:

1. Tworzymy procedurę, której wykonanie wymaga uprawnienia `ALTER TABLE` i która dodatkowo wyświetla informacje o tożsamościach, jakimi może się posłużyć:

```
CREATE PROC udpTrTbl AS
SELECT name,type
FROM sys.user_token;
TRUNCATE TABLE dbo.Tab1
```

2. Wiemy już, że nadanie użytkownikowi prawa do wykonania procedury nie wystarczy. Tym razem przekonamy się dodatkowo, jakimi uprawnieniami mogła posłużyć się procedura:

```
GRANT EXECUTE ON udpTrTbl
TO Szelor;

EXEC AS USER = 'Szelor'
EXEC udpTrTbl;
---------------------------------------
Szelor   SQL USER
public   ROLE
(2 row(s) affected)
Msg 1088, Level 16, State 7, Procedure udpTrTbl, Line 5
Cannot find the object "Tab1" because it does not exist or you do not have
➥permissions.
```

3. Ponieważ ani użytkownik `Szelor`, ani rola *public* nie miały wystarczających uprawnień do obcięcia tabeli, próba wykonania procedury skończyła się błędem. Nadajmy więc procedurze dodatkowe uprawnienia przez jej podpisanie:

```
REVERT;
CREATE CERTIFICATE CertSign
WITH SUBJECT = 'Podpisuje proc udpTrTbl';

ADD SIGNATURE TO udpTrTbl
BY CERTIFICATE CertSign;
```

4. Ponieważ nie można nadać uprawnień certyfikatom, na podstawie użytego do podpisania procedury certyfikatu utworzymy konto użytkownika i jemu nadamy wymagane do obcięcia tabeli uprawnienie:

```
CREATE USER uCertSign
FROM CERTIFICATE CertSign;

GRANT ALTER ON dbo.Tab1
TO uCertSign;
```

5. Jeżeli teraz użytkownik wykona procedurę, posłuży się ona uprawnieniami użytkownika uCertSign i tabela zostanie obcięta:

```
EXEC AS USER = 'Szelor'
EXEC udpTrTbl;

REVERT;
--------------------------------------
Szelor          SQL USER
public          ROLE
uCertSign       USER MAPPED TO CERTIFICATE
(3 row(s) affected)
```

Szyfrowanie baz danych (dotyczy edycji Enterprise)

Edycja Enterprise serwera SQL 2008 pozwala szyfrować całe bazy danych w sposób niewidoczny dla użytkowników i aplikacji klienckich. **Po jej włączeniu automatycznie szyfrowane są wszystkie, z wyjątkiem danych `FILESTREAM`, zapisywane na dyskach informacje**. Oznacza to, że oprócz plików bazy danych zaszyfrowane zostaną: baza tempdb, kopie zapasowe (ich odtworzenie bez klucza DEK będzie niemożliwe) i migawki bazy danych.

> Szyfrowanie baz danych jest niezgodne z wprowadzoną w poprzedniej wersji serwera SQL szybką inicjalizacją plików (funkcją pozwalającą szybko powiększać pliki bazodanowe bez wcześniejszego zerowania miejsca na dysku).

Zasada działania

Szyfrowanie baz danych przebiega następująco:

1. Po włączeniu szyfrowania osobne procesy po kolei odczytują, szyfrują i zapisują wszystkie strony bazy danych[19]. Po zaszyfrowaniu strony dodawana jest do niej suma kontrolna.
2. Podczas odczytywania strony sprawdzana jest jej suma kontrolna. Jeżeli jest ona poprawna, strona zostaje wczytana do bufora[20].
3. Przed zapisaniem w buforze strona zostaje odszyfrowana. Ponieważ wszystkie operacje na danych przeprowadzane są w buforach (a nie zapisanych na dysku stronach), zaszyfrowanie bazy nie ma żadnego wpływu na działanie aplikacji klienckiej.
4. Przed zapisaniem w pliku strona jest szyfrowana, a następnie dodawana jest do niej suma kontrolna.

[19] Dane w plikach *.mdf* i *.ndf* zorganizowane są w strony 8-kilobajtowe (ang. *Page*), które są też jednostką odczytu i zapisu.

[20] Bufor to ramka na wczytaną do pamięci stronę.

Konfiguracja

Żeby sprawdzić, czy informacje w wybranej bazie danych rzeczywiście zostaną zaszyfrowane, utworzymy w niej dodatkową tabelę i zapiszemy w niej przykładowe dane:

```
USE r07
CREATE TABLE Klienci (
 ID INT IDENTITY,
 Nazwisko VARCHAR (100) NOT NULL,
 NumerKarty CHAR (16) NOT NULL);
INSERT INTO dbo.Klienci
VALUES ('Kowalski', '4444555566667777'),
       ('Nowak', '4444888899990000');
```

A następnie wykonamy kopię zapasową tej bazy i odtworzymy ją pod zmienioną nazwą:

```
BACKUP DATABASE r07 TO
DISK = 'C:\Temp\r07.bak'
WITH INIT;
GO
RESTORE DATABASE r07Zaszyfrowana
FROM DISK = N'C:\Temp\r07.bak' WITH
MOVE 'r07' TO N'C:\Temp\r07Zaszyfrowana.mdf',
MOVE 'r07_Log' TO N'C:\Temp\r07Zaszyfrowana.ldf',
REPLACE;
```

Do szyfrowania baz danych używany jest klucz DEK, który z kolei musi być chroniony certyfikatem wystawionym w bazie master:

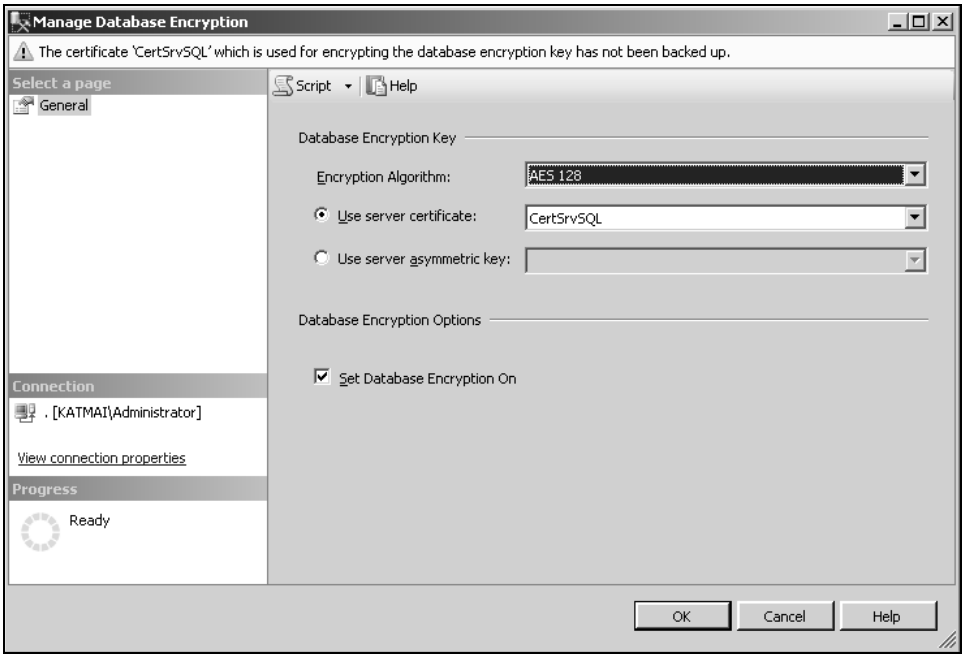
```
USE master;
CREATE MASTER KEY ENCRYPTION BY
PASSWORD = 'T@jneHasl0!';
CREATE CERTIFICATE CertSrvSQL
WITH SUBJECT = 'Certyfikat TDE';
```

Zaszyfrować, odszyfrować czy zmienić użyty do tego klucz możemy z pomocą konsoli SSMS. Żeby zaszyfrować bazę r07Zaszyfrowana:

1. Kliknij prawym przyciskiem myszki w oknie eksploratora obiektów bazę *r07Zaszyfrowana* i z menu kontekstowego wybierz *Tasks/Manage Database Encryption*.
2. Po wybraniu użytego do szyfrowania bazy algorytmu i wskazaniu certyfikatu zaznacz pole wyboru *Set Database Encryption On* i kliknij *OK* (rysunek 7.4).

Zaszyfrowanie tak małej bazy danych będzie prawie natychmiastowe. Po sprawdzeniu, czy zaszyfrowana baza danych jest dostępna w taki sam sposób jak oryginalna baza, wykonamy jej kopię i zatrzymamy serwer SQL — pozwoli to sprawdzić, czy zapisane w tej bazie informacje zostały zaszyfrowane[21]:

[21] Wartość 3 atrybutu `encryption_state` oznacza, że baza została zaszyfrowana, 2 — że szyfrowanie jest w toku.



Rysunek 7.4. *Raz skonfigurowane szyfrowanie bazy danych możemy też włączać i wyłączać w oknie jej właściwości, na zakładce Options*

```
SELECT DB_NAME(database_id), encryption_state, key_algorithm, key_length
FROM sys.dm_database_encryption_keys;
----------------------------------------
tempdb   3      AES     128
r07      3      AES     128

SELECT B1.Nazwisko, B2.NumerKarty
FROM r07.dbo.Klienci AS B1 INNER JOIN r07Zaszyfrowana.dbo.Klienci AS B2
ON B1.ID = B2.ID;
----------------------------------------
Kowalski 4444555566667777
Nowak    4444888899990000

BACKUP DATABASE r07Zaszyfrowana
TO DISK = 'C:\Temp\r07Zaszyfrowana.bak'
WITH INIT;

SHUTDOWN;
```

Żeby sprawdzić, czy dane rzeczywiście zostały zaszyfrowane:

1. Otwórz w programie WordPad plik *C:\Program Files\Microsoft SQL Server*
   ↪MSSQL10.MSSQLSERVER\MSSQL\DATA\r07.mdf (plik niezaszyfrowanej bazy) i poszukaj w nim nazwiska Kowalski. Zostanie ono znalezione, razem z numerem karty kredytowej. Jeżeli jednak w ten sam sposób przeszukamy plik *C:\Temp\r07zaszyfrowana.mdf*, nie znajdziemy w nim ani nazwisk, ani numerów kart kredytowych.

2. Otwórz w tym samym programie plik kopii zapasowej niezaszyfrowanej i zaszyfrowanej bazy (pliki *C:\Temp\r07.bak* oraz *C:\Temp\r07Zaszyfrowana.bak*) i sprawdź, czy można w nich znaleźć nazwiska. W pierwszym przypadku zostaną one znalezione, w drugim — nie.
3. Uruchom serwer SQL, klikając prawym przyciskiem myszy jego nazwę w oknie eksploratora obiektów konsoli SSMS i wybierając z menu kontekstowego zadanie *Start*.

Przenoszenie zaszyfrowanych baz danych

Atakujący może przenieść bazę i odtworzyć ją na własnym serwerze na dwa sposoby: poprzez skopiowanie plików bazodanowych i podłączenie ich na docelowym serwerze lub tworząc i odtwarzając kopię zapasową bazy. W obu przypadkach do odszyfrowania bazy (czy to pliku danych i dziennika, czy pliku kopii zapasowej) będzie jednak potrzebował klucza DEK. Odszyfrowanie tego klucza jest jednak niemożliwe bez certyfikatu znajdującego się w bazie master oryginalnego serwera SQL.

Odtworzenie zaszyfrowanej bazy danych wymaga wcześniejszego odtworzenia na docelowym serwerze chroniącego klucz DEK certyfikatu wraz z odpowiadającym mu kluczem prywatnym:

```
RESTORE DATABASE r07Zaszyfrowana
FROM DISK = 'C:\Temp\ r07Zaszyfrowana.bak'
WITH MOVE 'r07' TO 'C:\Temp\Kopia.mdf',
MOVE 'r07_Log' TO 'C:\Temp\Kopia.ldf',
REPLACE;
---------------------------------------
Msg 33111, Level 16, State 3, Line 1
Cannot find server certificate with thumbprint
'0xD726F9B73062701FAA832C1D3A200A4B7EB0E82A'.
Msg 3013, Level 16, State 1, Line 1
RESTORE DATABASE is terminating abnormally.
```

Musimy więc najpierw wyeksportować certyfikat i związany z nim klucz prywatny do plików:

```
USE master;
BACKUP CERTIFICATE CertSrvSQL
TO FILE = 'C:\Temp\CertSrvSQL'
WITH PRIVATE KEY (FILE = 'C:\Temp\CertSrvSQLKey',
 ENCRYPTION BY PASSWORD = 'Siln3h@$l0');
```

A następnie odtworzyć je na docelowym serwerze. Jeżeli w bazie master tego serwera nie został wcześniej wystawiony klucz DMK, będziemy musieli wcześniej go utworzyć:

```
USE master
CREATE MASTER KEY ENCRYPTION BY
PASSWORD = 'InnE@leTezS1lneH@sl0';

CREATE CERTIFICATE CertSrvSQL
FROM FILE = 'C:\Temp\CertSrvSQL'
WITH PRIVATE KEY (FILE = 'C:\Temp\CertSrvSQLKey',
  DECRYPTION BY PASSWORD = 'Siln3h@$l0');
```

Teraz możliwe już będzie odtworzenie kopii zapasowej zaszyfrowanej bazy.

Wydajność

Zastosowany sposób szyfrowania baz danych ma jeszcze jedną zaletę — **jeżeli odczytywane lub modyfikowane dane znajdują się już w pamięci, dostęp do nich będzie równie szybki jak do niezaszyfrowanych danych.** Ponieważ to podsystem wejścia-wyjścia, a nie procesor, jest wąskim gardłem większości serwerów, szyfrowanie baz danych powoduje niewielkie (z reguły nieprzekraczające 5%) spadki wydajności serwera mierzone czasem jego odpowiedzi. Szyfrowanie jednak znacznie zwiększa obciążenie procesora podczas odczytywania i zapisywania danych.

Monitorowanie i wykrywanie włamań

Zabezpieczenia mają utrudnić atak poprzez zablokowanie automatycznych prób włamania, zniechęcenie atakujących i ukrycie słabych punktów systemu. Nie zagwarantują jednak jego bezpieczeństwa. Mogą natomiast wydłużyć czas potrzebny na zaatakowanie systemu i zmusić atakujących do przeprowadzania łatwych do zauważenia operacji. Dlatego każda strategia zabezpieczeń musi obejmować monitorowanie, czyli zbieranie i regularne analizowanie danych na temat działania serwera i aktywności jego użytkowników.

Dzienniki serwera SQL

Do zbierania danych o serwerze możemy wykorzystać dzienniki zdarzeń serwera SQL 2008. Domyślnie zawierają one tylko informacje o nieudanych próbach zalogowania. Możemy monitorować zarówno udane, jak i nieudane próby nawiązania połączenia, zmieniając opcję *Login auditing* dostępną w oknie właściwości serwera na zakładce *Security*.

Do analizy zebranych w ten sposób danych możemy użyć dostępnego na witrynie Microsoft Download programu Log Parser albo systemowego narzędzia wiersza polecenia `FindStr`.

Plik śledzenia

Informacje na temat aktywność użytkowników można zapisać w pliku śledzenia. Najprościej jest przygotować szablon instrukcji w programie Profiler. Po wybraniu pustego szablonu i opcji zapisania danych do pliku:

1. Wybierz przechwytywane zdarzenia z kategorii *Security Audit* (np. *Audit Login*, *Audit Logout*, *Audit Add Login to Server Role Event* i *Audit Change Audit Events*).
2. Ewentualnie ogranicz i przefiltruj zapisywane informacje.

3. Uruchom śledzenie, następnie zatrzymaj je i z menu *File* wybierz opcję *Export/Script Trace Definition/For SQL Server 2005-2008*. Wygenerowany w ten sposób skrypt *.sql* należy wykonać w bazie master.
4. Do analizowania zapisanych w pliku śledzenia informacji użyj funkcji tabelarycznej `fn_trace_gettable`.

Wyzwalacze

Wyzwalacz (ang. *Trigger*) to specjalny typ procedury składowanej powiązanej z wybranym obiektem bazodanowym (tabelą, bazą danych lub całym serwerem SQL) i automatycznie wywoływanej wykonaną na tym obiekcie operacją użytkownika.

> Wyzwalacze są wykonywane synchronicznie w ramach transakcji użytkownika; jeżeli nawet użytkownik nie rozpoczął jawnie transakcji, kod wyzwalacza będzie wykonany w ramach tej transakcji, co oznacza, że wyzwalacz może wycofać instrukcje użytkownika[22] oraz że do momentu zakończenia jego działania serwer SQL 2008 utrzyma blokady założone przez oryginalną instrukcję.

Wyzwalacze DML

Wyzwalacze DML wywoływane są w momencie wstawiania, usuwania lub modyfikowania danych w powiązanych z nimi tabelach lub widokach. Ich wywołanie ma miejsce tylko raz dla instrukcji użytkownika, a nie dla każdego wiersza zmodyfikowanego w ramach tej instrukcji osobno.

> Monitorowanie aktywności użytkowników poprzez wyzwalacze DML może mieć bardzo niekorzystny wpływ na wydajność serwera.

Jeżeli już decydujemy się na używanie wyzwalaczy DML do monitorowania, powinniśmy wybrać wyzwalacze typu `INSTEAD OF` — **takie wyzwalacze uruchamiane są zamiast, a nie po instrukcjach użytkownika**. Dzięki temu, że wyzwalacze są wykonywane w ramach transakcji użytkownika, mają one dostęp do nowej (proponowanej) oraz poprzedniej wersji danych — pierwsze z nich znajdują się w wirtualnej tabeli `Inserted`, drugie w wirtualnej tabeli `Deleted`. Na przykład, żeby zapisywać w tabeli *Klienci* informacje o tym, kto i kiedy ostatnio zmodyfikował jej zawartość:

1. Dodaj do tej tabeli dwie kolumny:

```
USE r07
ALTER TABLE dbo.Klienci
ADD Kto VARCHAR(200), Kiedy DATETIME;
```

[22] Oznacza to również, że jeżeli wyzwalacz zapisze jakieś dane w tabeli, a następnie wycofa transakcję, zapisane przez niego dane także zostaną skasowane.

2. Utwórz wyzwalacz, który będzie wykonany zamiast oryginalnej instrukcji użytkownika i który dodatkowo zmodyfikuje wartości pól `Kto` i `Kiedy`:

```
CREATE TRIGGER tAudit
ON dbo.Klienci INSTEAD OF UPDATE AS
BEGIN
 IF (@@ROWCOUNT>0)
 UPDATE dbo.Klienci
 SET Nazwisko = I.Nazwisko,
 NumerKarty = I.NumerKarty,
 Kto = SYSTEM_USER + ' ' + USER_NAME(),
 Kiedy = CURRENT_TIMESTAMP
 FROM dbo.Klienci AS K JOIN INSERTED AS I
 ON K.ID = I.ID
END;
```

3. Przetestuj działanie wyzwalacza, modyfikując nazwisko jednego z klientów i odczytując zawartość całej tabeli:

```
UPDATE dbo.Klienci
SET Nazwisko = 'Majewski'
WHERE ID=1;

SELECT *
FROM dbo.Klienci;
----------------------------------------
1 Majewski    4444555566667777  KATMAI\Administrator dbo   2008-07-16 14:42:27.537
2 Nowak       4444888899990000  NULL                       NULL
```

Wyzwalacze DDL

Wyzwalacze DDL są uruchamiane w momencie wykonywania przez użytkownika instrukcji `CREATE`, `ALTER`, `DROP`, `GRANT`, `DENY`, `REVOKE` lub `UPDATE STATISTICS`. Wyzwalacze tego typu mogą być tworzone na poziomie serwera lub bazy danych.

Informacje o zdarzeniu, które spowodowało wywołanie wyzwalacza, zwraca funkcja `EventData()`. Funkcja ta może wystąpić tylko w ramach wyzwalacza DDL, a struktura wynikowego dokumentu XML zależy od typu przechwyconej instrukcji.

W poniższym przykładzie pokazujemy, jak utworzyć wyzwalacz, którego działanie polega na zapisywaniu w tabeli dziennika informacji o wskazanych, wykonywanych przez użytkowników instrukcjach DDL:

```
CREATE TABLE master.dbo.logDDL
(id INT IDENTITY PRIMARY KEY,
uzytkownik SYSNAME,
czas DATETIME,
instrukcja SYSNAME,
obiekt SYSNAME,
opis XML);
GO
CREATE TRIGGER trLogDDL
ON ALL SERVER
FOR CREATE_DATABASE,ALTER_DATABASE, DROP_DATABASE,DDL_LOGIN_EVENTS
AS
```

```
DECLARE @doc AS XML
SET @doc = EVENTDATA()
INSERT INTO master.dbo.logDDL (uzytkownik, czas,instrukcja, obiekt, opis)
VALUES (
CAST(@doc.query ('data(//LoginName)') AS SYSNAME),
CAST(@doc.query ('data(//PostTime)') AS VARCHAR(23)),
CAST(@doc.query ('data(//EventType)') AS SYSNAME),
CAST(@doc.query ('data(//ObjectName)') AS SYSNAME),
@doc );
```

Żeby sprawdzić działanie wyzwalacza, wykonaj poniższe instrukcje:

```
CREATE DATABASE r07Test;
DROP DATABASE r07Test;
ALTER LOGIN Admin WITH PASSWORD='dusza';

SELECT uzytkownik, czas, instrukcja, obiekt
FROM master.dbo.logDDL;
---------------------------------------
KATMAI\Administrator    2008-07-16 14:52:41.730      CREATE_DATABASE
KATMAI\Administrator    2008-07-16 14:52:41.800      DROP_DATABASE
KATMAI\Administrator    2008-07-16 14:52:41.810      ALTER_LOGIN    Admin
```

Wyzwalacze logowania

Wyzwalacze logowania są wywoływane w momencie próby zalogowania się użytkownika do serwera SQL 2008. Tak jak pozostałe typy wyzwalaczy, umożliwiają one wycofanie transakcji, co w ich przypadku oznacza odrzucenie próby zalogowania. Informacje na temat przechwyconej próby zalogowania zwraca funkcja `EVENTDATA()`.

Szczegółowy opis wyzwalaczy logowania wraz z przykładem ich zastosowania do monitorowania użytkowników zawiera dostępny na polskiej witrynie Microsoft TechNet artykuł *SQL Server 2005 Service Pack 2 przedstawia — Logon Trigger*.

Monitorowanie wszystkich operacji (dotyczy edycji Enterprise)

Przedstawione do tej pory techniki monitorowania miały dwie podstawowe wady:

1. **Nie są zintegrowane z powszechnie używanym do monitorowania aktywności dziennikiem systemu Windows**. Oznaczało to, że administratorzy muszą odczytywać dane w różnych formatach, korzystając w tym celu z kilku różnych, nie zawsze znanych sobie narzędzi.
2. **Nie pozwalają definiować zasad inspekcji równie łatwo jak w przypadku obiektów systemu Windows**. Większość administratorów jest przyzwyczajona do określania, czyje próby (przeprowadzone przez jakich użytkowników) odwołania się do wybranych obiektów mają być śledzone.

Oba te problemy rozwiązała dostępna wyłącznie w edycji Enterprise funkcja monitorowania wszystkich zdarzeń. Działa ona następująco:

1. Serwer SQL 2008 poprzez mechanizm rozszerzonych zdarzeń[23] zgłasza monitorowane zdarzenie.
2. Zdarzenia, które mają być odnotowane, są konwertowane do postaci rekordów audytu. Chociaż nie wszystkie zdarzenia zgłaszają komplet informacji, to rekordy audytu mają jeden, zawsze taki sam schemat.
3. Rekordy audytu są zapisywane we wskazanej lokalizacji: w pliku binarnym, systemowym dzienniku aplikacji lub w systemowym dzienniku zabezpieczeń.


Wskazówka

O ile tylko rekordy audytu zapisywane są asynchronicznie, kluczowy dla wydajności tego rozwiązania jest czas zgłoszenia zdarzenia. Wewnętrzne testy firmy Microsoft pokazują, że na dwuprocesorowym komputerze PIV z zegarem 2 GHz nie przekracza on 2 mikrosekund. W rezultacie monitorowanie wszystkich zdarzeń jest bardziej wydajne niż pliki śledzenia i w większości przypadków nie ma widocznego wpływu na wydajność serwera SQL.

Konfigurując audyt, należy określić:

1. Jego nazwę (audyt jest nowym typem obiektów poziomu serwera SQL 2008).
2. Maksymalne opóźnienie, z jakim rekordy audytu będą zapisywane (domyślnie są one zapisywane asynchronicznie z opóźnieniem do 1 sekundy, zmniejszając opóźnienie do 0, wymusimy synchroniczne monitorowanie zdarzeń, co może mieć niekorzystny wpływ na wydajność serwera[24]).
3. Czy w przypadku braku możliwości odnotowania zdarzenia serwer SQL 2008 ma być automatycznie wyłączony.
4. Lokalizację — rekordy audytu mogą być zapisywane w pliku, systemowym dzienniku aplikacji lub systemowym dzienniku zabezpieczeń (rysunek 7.5).

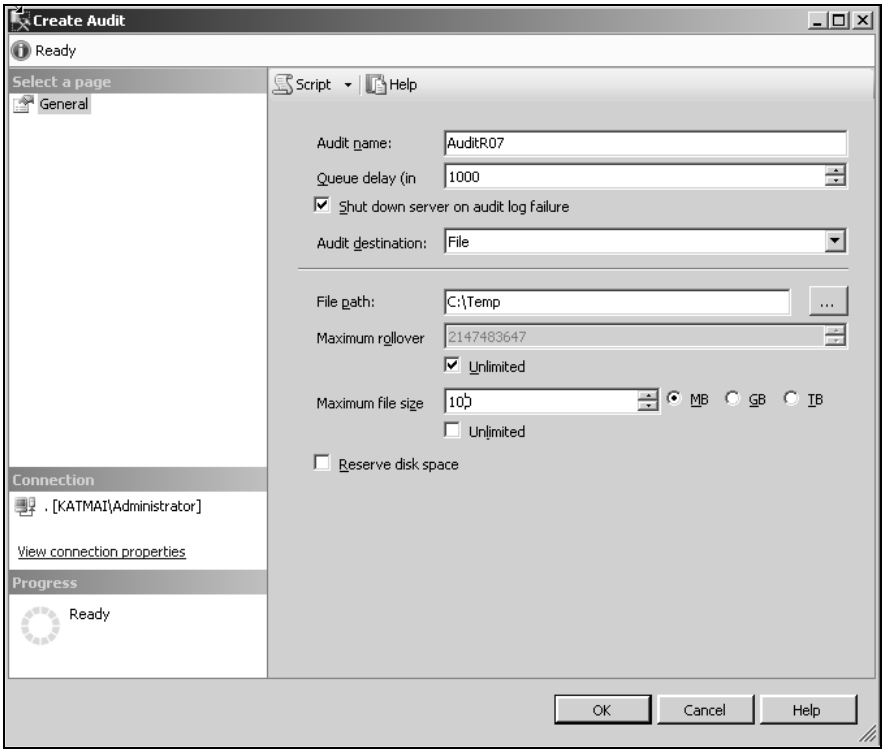

Wskazówka

Zapis w dzienniku zabezpieczeń jest niemożliwy w systemie Windows XP. W pozostałych systemach wymaga on nadania usłudze serwera SQL uprawnienia *Generate security audits* (wbudowane konta LOCAL SERVICE i NETWORK SERVICE mają to uprawnienie) oraz włączenia (za pomocą konsoli MMC *Lokalne zasady zabezpieczeń* lub narzędzia *auditpol.exe*) udanych i nieudanych inspekcji zdarzeń dostępu do obiektów.

Następnie należy określić specyfikację audytu na poziomie serwera SQL 2008 i wybranej bazy danych — z audytem można powiązać tylko jedną specyfikację poziomu serwera i jedną specyfikację poziomu bazy danych. **Specyfikacje zawierają informacje o tym, jakie zdarzenia mają być monitorowane**. Monitorowane mogą być zdarzenia zachodzące na poziomie serwera SQL (np. nieudane próby zalogowania czy wykonanie instrukcji `EXECUTE AS LOGIN`) oraz pojedynczych baz danych (np. skasowanie

[23] Opis mechanizmu rozszerzonych zdarzeń znajduje się w rozdziale 5.

[24] Maksymalne opóźnienie może wynosić prawie 25 dni.



Rysunek 7.5. *Audyt można utworzyć, rozwijając w eksploratorze obiektów sekcję Security, klikając prawym przyciskiem myszy folder Audit i wybierając opcję New Audit*

danych ze wskazanej tabeli czy wykonanie instrukcji `EXECUTE AS USER`). Tworzenie specyfikacji audytu ułatwiają predefiniowane grupy zdarzeń i możliwość definiowania poprzez konsolę SSMS:

1. Na poziomie serwera można monitorować m.in. grupy zdarzeń *SUCCESSFUL_LOGIN_GROUP*, *FAILED_LOGIN_GROUP*, *LOGOUT_GROUP*, *DBCC* czy *AUDIT_CHANGE_GROUP*. Poniższa instrukcja tworzy specyfikację audytu poziomu serwera monitorującą wykonanie instrukcji i klauzuli `EXECUTE AS LOGIN`:

```
USE master
CREATE SERVER AUDIT SPECIFICATION ServerAuditSpec
FOR SERVER AUDIT AuditR07
ADD (SERVER_PRINCIPAL_IMPERSONATION_GROUP)
WITH (STATE = ON);
```

2. Na poziomie bazy danych można monitorować m.in. grupy zdarzeń *DATABASE_PERMISSION_CHANGE_GROUP*, *SCHEMA_OBJECT_*
↪*CHANGE_GROUP*, *DATABASE_PRINCIPAL_IMPERSONATION_GROUP* oraz pojedyncze akcje wskazanych użytkowników, takie jak wykonanie przez nich instrukcji `SELECT`, `UPDATE`, `INSERT`, `DELETE`, `EXECUTE` czy `RECEIVE`. Na przykład poniższa specyfikacja spowoduje odnotowywanie prób dostępu użytkownika `Szelor` do tabeli `Klienci`:

```
USE r07
CREATE DATABASE AUDIT SPECIFICATION [DatabaseAuditSpecR07]
FOR SERVER AUDIT AuditR07
ADD (SELECT, INSERT, UPDATE ON dbo.Klienci BY Szelor)
WITH (STATE = ON);
```

Zanim włączymy audyt, nadamy jeszcze użytkownikowi `Szelor` uprawienia do odczytywania i wstawiania wierszy do tabeli `Klienci`:

```
GRANT SELECT, INSERT ON dbo.Klienci TO Szelor;

USE master
ALTER SERVER AUDIT AuditR07
WITH (STATE = ON);
```

Informacje na temat audytu znajdziemy w trzech dynamicznych widokach:

1. Widok `sys.dm_audit_actions` zwraca listę wszystkich możliwych do monitorowania akcji.
2. Widok `sys.dm_server_audit_status` zwraca szczegółowe informacje o zdefiniowanych dla serwera audytach.
3. Widok `sys.dm_audit_class_type_map` jest słownikiem używanych przy zapisie rekordów audytu skrótów.

Żeby zasymulować aktywność użytkowników wykonamy poniższe instrukcje:

```
USE r07;
GO
EXECUTE AS LOGIN = 'Marcin';
SELECT USER_NAME ();
GO
SELECT * FROM dbo.Klienci;
GO
INSERT INTO dbo.Klienci (Nazwisko,NumerKarty)
VALUES ('Nowak', '134-124-642-134');
GO
UPDATE dbo.Klienci
SET Nazwisko = ' Kowalski'
WHERE Nazwisko = 'Nowak';
```

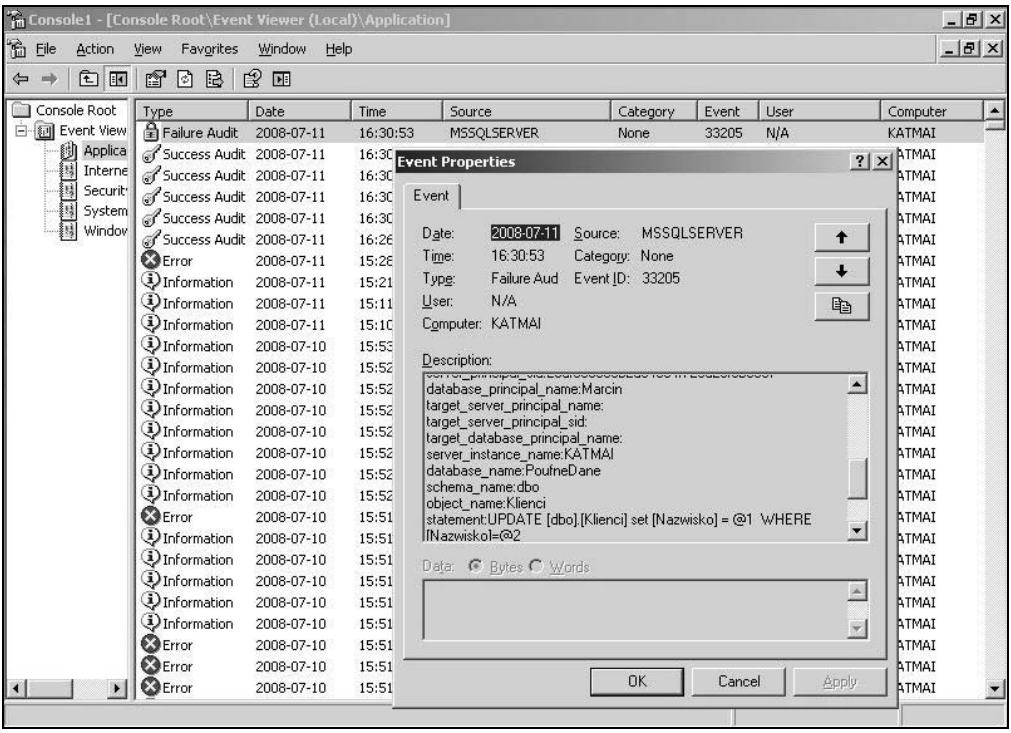
I sprawdzimy, czy informacje o nich trafiły do pliku audytu. Plik audytu ma format binarny, ale serwer SQL 2008 zawiera funkcję tabelaryczną umożliwiającą jego odczytanie (dla poprawy czytelności z ponad 30 kolumn zwracanych przez funkcję `fn_get_audit_file` wybraliśmy cztery i dodatkowo skorzystaliśmy ze słownika `sys.dm_audit_class_type_map`):

```
REVERT;
SELECT AuditFile.action_id, AuditFile.succeeded,AuditFile.database_principal_name,
➥Dict.class_type_desc , AuditFile.statement
FROM sys.dm_server_audit_status AS AuditStatus
CROSS APPLY sys.fn_get_audit_file (
 AuditStatus.audit_file_path, default, default) AS AuditFile
INNER JOIN sys.dm_audit_class_type_map AS Dict
ON Dict.class_type = AuditFile.class_type;
----------------------------------------
```

```
AUSC     1              SERVER AUDIT
IMP      1      dbo     SQL LOGIN      EXECUTE AS LOGIN = 'Szelor';
SL       1      Szelor  TABLE          SELECT * FROM dbo.Klienci;
IN       1      Szelor  TABLE          INSERT INTO
[dbo].[Klienci]([Nazwisko],[NumerKarty]) values(@1,@2)
SL       1      Szelor  TABLE          UPDATE [dbo].[Klienci] set [Nazwisko] = @1
↪WHERE [Nazwisko]=@2
```

Zmiana lokalizacji audytu wymaga jego zatrzymania i ponownego uruchomienia. Po wykonaniu poniższych instrukcji `ALTER SERVER AUDIT` rekordy audytu będą zapisywane w systemowym dzienniku aplikacji (rysunek 7.6):

```
USE master
ALTER SERVER AUDIT AuditR07 WITH (STATE = OFF);
ALTER SERVER AUDIT AuditR07 TO APPLICATION_LOG;
ALTER SERVER AUDIT AuditR07 WITH (STATE = ON);
```



Rysunek 7.6. *Dziennik aplikacji systemu Windows z wpisami inspekcji pochodzącymi z audytu serwera SQL*