

# Problemy SQL Injection w bazach danych Oracle

Michał Agata

*Instytut Mechatroniki i Systemów Informatycznych Politechniki Łódzkiej*

*e-mail: anubis@agm.pl*

**Promotor: prof. nadzw. Politechniki Łódzkiej, dr hab. inż. Adam Pelikant**

## **Abstrakt**

Metody SQL Inject są coraz większym zagrożeniem dla bezpieczeństwa informacji przechowanych w bazach danych Oracle. Metody te są coraz regularniej omawiane na listach dyskusyjnych związanych z bezpieczeństwem, forach czy konferencjach. Jak dotąd napisano wiele dobrych prac na temat SQL Injection i parę o bezpieczeństwie bazy danych Oracle, ale niewiele z nich opisywało połączenie techniki SQL Inject

w zależności od oprogramowywania bazy danych. Ten artykuł postara się położyć nacisk na badanie metod SQL Injection oraz metodykę ataków na bazę danych Oracle. Celem artykułu jest uchronienie użytkowników Oracle od niebezpieczeństw związanych z SQL Inject'em i sugerować proste drogi obrony przed atakami tego typu.

Oracle jest olbrzymim i skomplikowanym produktem. SQL Inject może być stosowany do wielu z jego modułów, języków i API, tak więc artykuł jest wprowadzeniem do głębszych badań problemu. Postaram się pokazać główne trendy technik SQL Inject, omówić je i wskazać rozwiązania, które mogą pomóc w zabezpieczeniu bazy danych Oracle.

## Czym jest SQL Injection?

SQL Injection to sposób ataku na dane znajdujące się w bazie – nawet, kiedy sam serwer jest chroniony zaporą ogniową (FIREWALL). Jest to metoda, dzięki której parametry aplikacji sieciowej są zmodyfikowane tak, aby zmienić deklaracje SQL, które przechodzą do bazy danych aby otrzymać dane w postaci odpowiedzi. Na przykład, przez dodawanie jednego apostrofu (') do listy parametrów, można spowodować wykonanie drugiego zapytania zagnieżdżonego w pierwszym.

Atak na bazę danych używając metody SQL Injection, może być motywowany przez dwa główne założenia:

1. Ukraść dane z bazy danych (którego to dane w sposób jawny nie powinny być dostępne) lub uzyskać dane odnośnie konfiguracji systemu (których znajomość zezwoliłaby na sporządzenie profilu późniejszego ataku). Przykładem może być atak mający na celu pozyskanie hash'y haseł bazy danych, które w późniejszym czasie mogą być złamane metodą siłową (brute-force).
2. Dostać się do wnętrza struktury organizacji poprzez komputery, na których zainstalowana jest działająca baza danych. Cel może zostać osiągnięty poprzez użycie pakietów procedur i rozszerzeń języka 3GL, które pozwalają na dostęp O/S.

Jest wiele sposobów by wykorzystać metodę SQL Inject na bazie danych Oracle. Atak bazuje na użytym języku programowania oraz API. Poniżej prezentuję podstawowe języki

i narzędzia, mogące być częścią aplikacji sieciowej, mające jednocześnie możliwość łączenia się z bazą danych Oracle:

- JSP
- ASP
- XML, XSL i XSQL
- Javascript
- VB, MFC, i inne narzędzia oraz API bazujące na ODBC
- Raporty, typowe narzędzia Oracle
- Języki 3- i 4GL takie jak C, OCI, Pro C, COBOL
- Perl i skrypty CGI, które posiadają możliwość łączenia się z bazą danych Oracle

Każdy z powyższych może służyć jako baza do przeprowadzenia ataku SQL Inject na bazę danych Oracle. Do przeprowadzenia ataku, musi być jednak spełniony podstawowy warunek. Przede wszystkim, aby SQL Inject był możliwy do przeprowadzenia,

w aplikacji, produkcie czy narzędziu, musi być dozwolone użycie dynamicznego SQL.

Ostatnim ważnym punktem, który zazwyczaj nie jest omawiany to fakt, iż SQL Inject przeprowadzany wobec bazy danych, nie jest tylko problemem aplikacji sieciowej. Groźny kod SQL, używany przy ataku SQL Inject, może być wstrzyknięty w przypadku każdej aplikacji, która umożliwia użytkownikowi wprowadzanie danych oraz korzysta z dynamicznego SQL'a. Oczywiście, aplikacje sieciowe przedstawiają najwyższy poziom ryzyka, jako, że każdy użytkownik, posiadający przeglądarkę sieciową oraz połączenie internetowe, może być w stanie przeprowadzić atak SQL Inject.

Metody ochrony przez takimi atakami zostaną omówione

w dalszej części artykułu, jednakże w tym miejscu chciałbym wspomnieć o kilku faktach. Dane osadzone w bazie danych Oracle, powinny być chronione przez dostępem użytkowników korporacji, pracowników korporacji czy użytkowników zewnętrznych, upoważnionych do pracy z aplikacjami, które operują na chronionych danych. Użytkownicy mogą oczywiście chcieć uzyskać dostęp do danych, które nie są dla nich przeznaczone (do wglądu, odczytu). Należy więc pamiętać,

że największe niebezpieczeństwo będzie pochodziło właśnie od uprawnionych użytkowników (pracowników, użytkowników „wewnętrznych”).

Obrona przez atakiem typu SQL Inject składa się głównie

z dwóch etapów. Są to:

1. Kontrola kodu źródłowego aplikacji i zapobieganie wszelkim próbom wstrzykiwania kodu SQL. (zapobieganie od strony programistycznej poprzez walidację i sprawdzanie czy na przykład – utworzenie logicznych limitów oraz zasad bezpieczeństwa)
2. Stosowanie zasady „najniższego dostępu”, wedle której wobec przeprowadzonego ataku SQL Inject, użytkownik jest w stanie ukraść NIE WIĘCEJ danych, niż projektant zezwala w przypadku poprawnej pracy aplikacji.

### **Czy SQL Injection może być wykryte?**

SQL Injection można wykryć, ale prawdopodobnie nie zawsze i nie w czasie rzeczywistym. Jest wiele skomplikowanych powodów:

- Wiele różnych form ataku SQL Inject, które mogą mają miejsce - są ograniczone tylko przez wyobraźnię hackera oraz dalekowzroczność DBA (a właściwie jej brak) – wobec ochrony i przestrzegania zasad dostępu do bazy danych
- Identyfikowanie kodu SQL, który nie powinien wystąpić, nie jest proste. SQL Inject jest możliwe poprzez możliwość użycia dynamicznego SQL w aplikacjach.

To implikuje fakt, iż zestaw poprawnych instrukcji SQL jest ciężki, jeśli nie niemożliwy do identyfikacji. Tak więc, jeśli poprawny zestaw deklaracji SQL jest niemożliwy do identyfikacji, podobnie niemożliwe jest zdefiniowanie „nielegalnego”

- Rozróżnianie napastnika od uprawnionego do wykonywania specjalnych operacji Administratora jest o tyle trudne, że atakujący może logować się jako Administrator bazy danych
- Wykrywanie SQL Inject wiązałoby się z parsowaniem całego polecenia SQL oraz podziałem go na grupy logiczne. Tym samym, na przykład nazwy perspektyw czy tabel, musiałyby być wydzielane z całości polecenia SQL i sprawdzane pod kątem możliwości (od strony bezpieczeństwa) wykonania zadeklarowanych dla nich operacji
- Użyte techniki nie powinny znacząco wpływać na wydajność bazy
- Podobnie jak przy parsowaniu polecenia SQL, w tym samym czasie należałoby parsować i wydobywać nazwy użytkowników oraz sygnatury czasowe (weryfikacja czasu zdarzenia, porównanie logów bazy danych z innymi obecnymi w systemie)

### **Gotowe rozwiązania**

Obecnie na rynku nie istnieją żadne gotowe rozwiązania, które na bieżąco blokowałyby możliwość ataku SQL Inject. Istnieje spora gama produktów, które w połączeniu z systemami firewall czy IDS, zachowują

się jak proxy dla Oracla, jednakże to rozwiązanie posiada wiele ograniczeń, nie jest elastyczne i nie zdaje egzaminu w sytuacjach, gdzie mamy do czynienia z dużym ruchem pomiędzy aplikacjami a samą bazą Oracle. W fazie projektu znajdują się moduły do takich produktów jak Snort, jednakże Snort sam w sobie jest w stanie analizować pakiety danych, których składanie oraz formowanie pełnej kwerendy może być nie tylko czasochłonne, ale także dość znacznie obciążać zasoby serwera. Dodatkowo, w przypadku wykrycia ataku SQL Inject (analiza zapytania) może okazać się niemożliwe zareagowanie „na czas”.

### Pomysły na rozpoznanie ataku SQL Injection

Analizując samą kwerendę SQL, która wysyłana jest w stronę serwera SQL, dobrym punktem rozpoznania mogą być następujące charakterystyczne dane zawarte w jej ciele:

- dodatkowy łącznik **union**, który powoduje wczytanie danych z odrębnej perspektywy lub tabeli
- dodatkowy łącznik **select**, który dodany jest wewnątrz struktury zapytania
- **warunek** implikujący zwrócenie wartości true w formie **prostego argumentu**, czyli np. ‘a’=’a’, 1=1...
- **nazwę wewnętrznego polecenia lub funkcji**, która nie powinna występować w kwerendach pochodzących z ZEWNAŁTRZ bazy danych (a funkcjonalnie występująca w przypadku administracji bazą czy wywołania z konsoli zarządzania bazą)
- zapytanie w odniesieniu do tabel **systemowych** lub **tabel wymagających autentykacji** użytkownika innego niż zdefiniowany w aplikacji domyślny dostęp
- zapytanie, w ciele którego znajdują się symbole pominięcia dalszego jego ciała, np. „- -,”
- zapytanie, które **generuje większą liczbę błędów** (paranoicznie można założyć zapytanie, które generuje jakiegokolwiek błędy – gdyż oryginalne w wersji ostatecznej, wyzwalone z poziomu aplikacji, nie powinno zawierać błędów) – np. błędy **złej ilości argumentów w funkcjach** lub **odwołań do nieistniejących tabel lub kolumn** w tabelach (lub błędów na skutek tworzenia kwerendy z łącznikiem union na tabele, która posiada różną liczbę kolumn od poprzedniej)

W całej jednakże analizie, powinniśmy położyć nacisk na prostotę oraz szybkość działania. Skomplikowane i zasobożerne rozwiązania nie bywają ani efektywne ani wydajne w swym działaniu. Paranoidalny poziom analizy (zbyt dokładna analiza kwerend i zbyt wysoki poziom rozpoznania) może doprowadzić do zatrzymania działania aplikacji na skutek blokady kwerend, które są generowane przez aplikację np. w przypadku większej liczby warunków nałożonych na formułowanie zapytania (najczęstszym przypadkiem są wielowarunkowe zapytania z modułów wyszukiwania danych). Nie należy stosować pojedynczych rozwiązań. Najlepiej proaktywnie analizować zapytania, rozwijać spektrum rozpoznania oraz używać w przy analizie wielu metod sprawdzania. Dobrym rozwiązaniem może być modularna budowa systemu analizy (np. wyniki wstępnej analizy przesyłane są do kolejnego modułu lub elementu analizy).

Powyższe punkty mogą stanowić bazę do stworzenia systemu wczesnego ostrzegania i analizy kwerendy pod kątem ewentualnego ataku SQL Inject. System rozpoznania (lub systemu w przypadku wielu technik rozpoznania) powinny móc:

- Przechwycić ciąg SQL tuż po wysłaniu go z aplikacji, zanim ciąg trafi do samej bazy Oracle (oczywiście pod względem wydajności procesu – najlepiej najszybciej jak to tylko możliwe)
- Zanalizować ciąg SQL pod kątem ewentualnego wystąpienia sygnatur ataku SQL Injection
- Przechwycić sygnatury czasowe oraz uwierzytelniające (wpisy odnośnie użytkownika)

Jeżeli zadbamy o spełnienie powyższych punktów, możemy pokusić się o umieszczenie ich w następujących programach lub stworzenie rozwiązań bazujących na:

- Istniejących snifferach aktywnych lub pasywnych
- Istniejących systemach IDS
- Programach analizujących pakiety w ruchu sieciowym
- Sieciowych plikach TRACE Oracle'a
- Serwerowych plikach TRACE Oracle'a
- Ekstrakcji ciągu SQL wprost ze zrzutu pamięci Oracle'a (SGA)
- Użycia dostępnych narzędzi Oracle typu LOG MINER i analizie plików LOG oraz obszarów zapisów systemowych
- Uruchomienia pełnego audytu Oracle
- Użycia triggerów bazy Oracle
- Użycia FGA (Fine Grained Audit)

Istnieje jednakże wiele ograniczeń wobec w/w metod lub pomysłów. Przede wszystkim powinniśmy być świadomi faktu, iż rozwiązanie polegające na dogłębnej analizie lub audycie, jest tylko i wyłącznie rozwiązaniem „nad rozlanym mlekiem”. Zwykle sytuacja ma miejsce, kiedy atak został zaaplikowany, zakończył się sukcesem, natomiast audytor stara się pozyskać maksymalnie dużo informacji odnośnie całości oraz zapobiec takowemu atakowi na przyszłość. Pozyskanie czy analiza pakietów sieciowych pomiędzy aplikacją i bazą Oracle czy też Farmie Serwerów z bazą Oracle może być niemożliwe. Bardzo często transmisja jest kodowana, co więcej, potrafi być kompresowana, tak więc pozyskanie pojedynczych pakietów, złożenie ich w całość, rozpakowanie oraz rozkodowanie transmisji, może być nie tylko czasochłonne ale i niemożliwe do zrealizowania w skończonej jednostce czasu. Metody przeszukiwania plików trace czy też zrzutów pamięci, nie tylko dramatycznie wyczerpują zasoby serwera, ale także są wysoce czasochłonne. Tak więc jedyną skuteczną metodą może być użycie rozwiązań, które pozwalają w czasie rzeczywistym na analizę logiczną ciągów SELECT. Jednakże, nawet w przypadku braku możliwości analizy czasu rzeczywistego ew. wystąpienia ataku SQL Inject, lepiej poświęcić zasoby i czas i dowiedzieć się o tym PO FAKCIE, niż w ogóle nie być świadomym zaistniałej sytuacji.

### **Krótkie omówienie poszczególnych możliwości analizy ataku**

#### **Sniffing pakietów**

W większości rozwiązań, sam ruch do bazy Oracle, nie jest upubliczniany. Istnieje jednakże wiele metod na przechwycenie pakietów nawet w takowym przypadku. W przypadku sieci, która oparta jest na zwykłych koncentratorach (HUB), przechwycenie pakietów jest dziecinnie proste (zasada działania zwykłych koncentratorów nie ogranicza ruchu pakietów na poszczególnych portach). Można zastosować zarówno DNS poisoning w przypadku wykrycia mnogości połączeń na sam serwer DNS

(ruch oparty na komunikacji polegającej na nazwach hostów, a nie bezpośrednich odwołań na zdefiniowane adresy IP), jak i ARP poisoning (celowe zatrucie tablic ARP w przypadku switch'ów). Można pokusić się także o bardziej eleganckie rozwiązanie (aby nie odcinać czasowo ruchu DO samego serwera z bazą Oracle) polegające na ataku MITM (Man in the Middle). Tym samym, przy wykorzystaniu odpowiednich metod (których w tym artykule omawiać nie będę) – stajemy się bastionem przekierowującym ruch do

serwera z bazą Oracle, jednakże cały ruch (przechodząc przez nasz komputer) może być przeanalizowany i sprawdzony.

Niestety, w przypadku ruchu kodowanego, możemy (bez użycia rozwiązań inżynierii wstecznej powiązanej z analizą sposobu kodowania oraz opracowania mechanizmów dekodowania transmisji) pokusić się o przechwytywanie co najwyżej ciągów niekodowanych ASCII, które następnie poddamy analizie. Do zalet takiego rozwiązania możemy na pewno zaliczyć fakt, iż w przypadku celowego rozmieszczenia MITM – system może być zaimplementowany na oddzielnym serwerze lub komputerze. Sama jednak analiza może być pamięciożerna oraz zasobożerna (w przypadku czasu procesora). Szczególnie, jeśli analizie poddamy wszystkie pakiety kierowane do serwera z bazą Oracle na pokładzie.

Kolejnym ograniczeniem i trudnością może być fakt, iż nasłuch powinien być prowadzony jak najbliżej źródła. Tak więc infrastruktura sieci powinna być odpowiednio rozwiązana w stosunku do proponowanego rozwiązania. Pod uwagę powinniśmy także wziąć fakt, iż w przypadku, kiedy ciąg ataku SQL Injection – jest argumentem funkcji (nie jest zawarty jako część oryginalnej kwerendy, ale podawany jako argument do wewnątrz funkcji na nim bazującej). W podobnym także przypadku, kiedy ciąg odwołuje się do wewnętrznej funkcji wewnątrz bazy danych, sam ciąg SQL nie będzie widoczny dla sniffera. Oczywiście pod uwagę powinniśmy wziąć także olbrzymią rotację i ruch pakietów, które są generowane przy współpracy aplikacji z bazą Oracle.

### **Log Miner**

Z poziomu Oracle mamy do dyspozycji dwie zasadnicze procedury umożliwiające przetrzaskanie logów bazy danych: DBMS\_LOGMNR oraz DBMS\_LOGMNR\_D. Procedury kolejno przetrzaskają logi bazy oraz logi generowane w trybie rzeczywistym (REDO). Logi REDO (generowane w trybie rzeczywistym) są tak naprawdę składnicą zapisków służących do ew. cofnięcia działania polecenia lub kwerendy (wykorzystywane chociażby w stosunku do cofania transakcji).

Istnieje wiele zalet zastosowania tego rozwiązania w przypadku wyszukiwania symptomów lub skutków ataku (nie zapominajmy, że tego typu analiza zazwyczaj prowadzona jest „po fakcie”, ze względu na wymagania czasowe oraz ilość danych, nie nadaje się do zastosowań czasu rzeczywistego). Najważniejszą zaletą jest możliwość przeniesienia plików z logami na zdalny komputer (np. o większej mocy obliczeniowej lub separatywny z punktu widzenia działania sieci). Dodatkowo, do analizy logów, udostępnione jest ze strony Oracle narzędzie graficzne OEM (Oracle Enterprise Manager).

Niestety, przy wykorzystaniu Log Miner’a, istnieje więcej wad niż omówionych wcześniej zalet. Przede wszystkim, w przypadku bazy MTS (Multi Threaded Server), narzędzie Log Miner nie może być użyte ze względu na wewnętrzną alokację pamięci samego narzędzia (narzędzie używa pamięci PGA, która nie jest widoczna dla wszystkich wątków serwera). Kolejnym ograniczeniem jest niewłaściwa współpraca z: połączonymi lub migrowanymi rzędami (rows), obiektami (objects), tablicami indeksów (index tables) czy klastrami (clusters). Dodatkowo, ciąg SQL utworzony przez Log Miner’a NIGDY nie jest w 100% zgodny z oryginalnym ciągiem użytym przez atakującego (spowodowane jest to specyfiką logów REDO).

Dodatkowo, aby w ogóle użyć narzędzia, baza musi mieć ustawiony tryb ARCHIVELOGMODE, natomiast zmienna `trans action_auditing` musi być ustawiona w stanie: `true`.

Pomimo wielu ograniczeń, Log Miner jest doskonałym narzędziem analizy, które może posłużyć do rozpoznania, zrozumienia a także w przyszłości – jako baza przeciw podobnym atakom SQL Injection.

### **Sniffing pakietów od strony Oracle**

Z poziomu narzędzia SQL Net (Oracle Networking) istnieje możliwość śledzenia transmisji. Pomocne mogą być następujące komponenty:

- Oracle names
- Connection Manager
- Names Control Utility
- Oracle Networking Client&Server

Oczywiście, można byłoby pokusić się o śledzeni transmisji od strony klienta, jednakże mogłoby to spowodować trudności w ogarnięciu całości rozwiązania. Największą wadą metody jest chyba rozmiar plików śledzenia (trace), które w stosunkowo krótkim okresie czasu potrafią osiągnąć znaczne rozmiary (w przypadku niewłaściwej administracji servera oraz samego procesu, doprowadzić do przepełnienia dysku). Dodatkowo, pomimo faktu, iż jesteśmy w stanie określić nazwę pliku oraz jego lokalizację, Oracle dodaje ze swojej strony ID procesu (PID) do nazwy pliku. Identyfikator NIE JEST znany do momentu podłączenia się użytkownika (a tym samym, wystartowania procesu SHADOW). Tak więc może wystąpić problem w rozpoznaniu aktywnie używanych plików, lokalizacji ewentualnego miejsca ataku czy ochrony przed przepełnieniem dysku. Jest to więc metoda stosunkowo mało wydajna.

### **Plik(i) śledzenia z punktu widzenia Oracle**

Istnieje także możliwość włączenia zrzutu stanu bazy Oracle (w trybie czasu rzeczywistego) z poziomu samej bazy danych (polecenia: ALTER <SYSTEM/SESSION> SQL\_TRACE=TRUE;). Rozwiązanie to również obarczone jest pewnymi ograniczeniami. Przede wszystkim, zrzut musiałby być włączony globalnie przez cały czas. Sam proces pochłania ogromne ilości zasobów komputera, na którym jest uruchamiany (im dłużej trwa sam proces, tym więcej zasobów jest zużywanych). Do analizy wystąpienia obcego ciągu (sygnatury ataku SQL Injection) potrzebny byłby zewnętrzny parser. Jako, że plik również zawierałby numer PID, ciężkie może być wyśledzenie aktualnie używanych plików i wykasowania starych, aby zapobiec przepełnieniu się dysku.

### **Oracle audit**

Przy analizie skutków czy sposobu ataku, można pokusić się o audyt wewnątrz bazy Oracle. Niestety, podstawowy audyt ma kilka dość poważnych ograniczeń: nie działa na poziomie rzędu (row), ciąg SQL w trybie rzeczywistym nie jest wychwytywany, dodatkowo – ciężko jest otrzymać rozpoznanie użycia ciągu SQL z łącznikiem **union**.

### **Database triggers**

Triggerzy wewnątrz bazy danych są kolejną linią obrony w przypadku użycia audytu. W przypadku użycia triggerów, pozbyć się możemy podstawowego ograniczenia od strony audytu (działanie na obiektach, nie na rzędach). Oczywiście w przypadku triggerów, musimy zadbać o oprogramowanie podstawowych procedur (INSERT, UPDATE, DELETE). Ograniczeniem triggerów jest niemożność użycia ich w przypadku wystąpienia kwerendy SELECT.

### **FGA (Fine Grained Auditing)**

Począwszy od wersji 9i, Oracle wprowadziło mechanizm FGA. Idea narzędzia polega na odpalaniu odpowiedniego triggera za każdym razem, kiedy wyrażenie (ciąg) SQL jest parsowany. Oczywiście, przy ustawianiu narzędzia, zobligowani jesteśmy do zdefiniowania polityk audytu (zasad – policies). Co więcej, narzędzie pozbawione jest uchybu standardowego audytu – może także monitorować zapytania typu SELECT. Reasumując – narzędzie FGA stanowi bazę do implementacji zewnętrznego systemu IDS.

### **Odczyt z SGA (System Global Area)**

Opis tej metody zostawiłem sobie na koniec (jako, z mojego punktu widzenia, najbardziej obiecującej metody wykrywania ataków typu SQL Injection). Najprostszym uzasadnieniem jest fakt, iż bazuje ona na „sercu” bazy danych. Wszystkie operacje SQL, które wykonywane są przez lub w bazie danych – przebiegają przez jakiś czas w SGA. Istnieją ograniczenia zastosowania tej metody, jednakże wydaje mi się, że przy umiejętnym stworzeniu silnika narzędzia, jest możliwe pozbycie się ograniczeń, które opiszę poniżej:

1. Odpytywanie wewnątrz SGA może zaowocować spadkiem wydajności (w przypadku nieumiejętnego zastosowania kwerendy – do całkowitego zablokowania systemu).
2. Ciąg SQL może być, najprościej mówiąc – przeoczony. Wiąże się to z faktem, iż przez SGA przełatują tysiące danych w jednostce czasu. Niektóre, na skutek olbrzymiej ilości danych płynących, mogą zostać z SGA zbyt szybko wyrzucone. Jednakże szansa na „znalezienie” obcego ciągu SQL jest stosunkowo wysoka.

## **PODSUMOWANIE I WNIOSKI**

Ataki typu SQL Injection, pomimo istnienia od jakiegoś czasu – są nadal nowością w przypadku opracowywania metod ich wykrywania. Jak szeroka jest skala zagrożenia – ciężko jest przewidywać. Składa się na to wiele czynników, głównym jednak jest fakt, iż wielu administratorów serwerów baz danych Oracle nie wie, iż takowy atak w ogóle został przeprowadzony wobec bazy danych (chyba, że skutki płynące z takowego ataku są wyjątkowo niezdarnie zamaskowane, a niekiedy – powodują problem w samej bazie danych).

Na spadek podatności bazy danych na ataki SQL Injection, można wpłynąć stosując się do kilku zasad:

- Starając się nie używać dynamicznego SQL. Jeśli użycie jest konieczne, w maksymalny możliwy sposób powinno być filtrowane jego użycie.
- Starając się nie używać dynamicznego PL/SQL gdziekolwiek w aplikacji. Jeśli nie można zastąpić takowego użycia, powinno stosować się zmienne typu **bind**.

- Starając się ograniczyć wyzwalanie komendy lub wykonanie ciągu SQL do zdefiniowanego statycznie użytkownika (aby nie wykonywały się one z prawami właściciela bazy danych).
- Starając się stosować zasadę „najniższego poziomu uprawnień”, czyli nadawania minimalnych uprawnień potrzebnych do wykonania danego zadania/czynności.

SQL Injection zyskuje coraz większe poparcie wśród potencjalnych napastników ze względu na klarowność i prostotę rozwiązania. Poza tym, po ataku istnieje (lub często nawet – nie istnieje) stosunkowo mało śladów, które mogłyby zostać zauważone i stanowić podstawę do wszczęcia alarmu lub kroków zapobiegania podobnym sytuacjom. Podstawową formą zabezpieczenia bazy danych przed atakami, oprócz ograniczania dynamicznego SQL, powinna być wyobraźnia oraz próba przewidzenia wszelkich możliwych uchybień podczas projektowania struktury bazy, oprogramowywania poleceń, funkcji czy samej aplikacji.