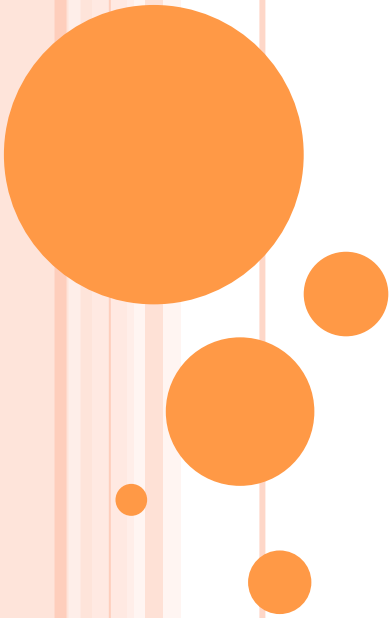

SQL INJECTION

by Fabrizio d'Amore
faculty of Ingegneria dell'Informazione
Università di Roma "La Sapienza"

WHAT IT IS, WHAT IT IS NOT

- capability of giving SQL commands to a *database engine* exploiting a pre-existing application
- not exclusive to Web applications, but widespread vulnerability in Web sites
  - vulnerabilities exist in 60% of Web sites they have tested (from: OWASP, Open Web Application Security Project)
- not due to inadequate development of Web applications, nor a fault of the Web / RDBMS server
  - developers not yet sufficiently aware
  - low-quality info in the Web on how to prevent the problem
  - detailed info in the Web on how to exploit vulnerabilities

WHAT APPLICATIONS ARE VULNERABLE?

- in practice, all databases based on SQL
  - MS SQL Server, Oracle, MySQL, Postgres, DB 2, Informix etc.

- databases accessed thru applications based on most of modern (and non-modern) technologies
  - Perl, CGI, ASP, PHP, XML, Javascript, VB, C, Java, Cobol etc.

HOW IT WORKS

- client injects SQL code into the input data of an application
  - typical scenario: application dynamically creates SQL query using altered data (obtained from outside), without good validation of such data
- target of the attack: server of an application
- goal: allow the client to access the database used by the attacked server

EXAMPLE

- if the following query returns data...

```
SELECT * FROM users
WHERE login = 'damore'
AND password = 'qwerty'
```

- example of login syntax ASP/MS SQL Server

```
var sql = "SELECT * FROM users WHERE login = '" +
formusr + "' AND password = '" + formpwd + "'";
```

- if
  - `formusr` = `' or 1=1 --`
  - `formpwd` arbitrary
- query becomes into

```
SELECT * FROM users
WHERE login = '' or 1=1
-- AND password = ...
```

SQL INJECTION ATTACK

- attacker can access database in read/write/admin
  - depends on the vulnerability of the specific DBMS
- impact of the attack is potentially HIGH

POSSIBLE HTML FORM

- from Wikipedia (http://it.wikipedia.org/wiki/SQL_injection)

```
<form action='login.php' method='post'>
  Username: <input type='text' name='user' />
  Password: <input type='password' name='pwd' />
  <input type='submit' value='Login' />
</form>
```

POSSIBLE LOGIN.PHP FILE

```
<?php
  //Prepares query, in a variable
  $query = "SELECT * FROM users WHERE user='".$_POST
  ['user']."' AND pwd='".$_POST['pwd']."'";
  //Execute query (suppose a valid connection to
  database is already open and its state is stored
  in $db)
  $sql = mysql_query($query,$db);
  //Count number of lines that have been found
  if(mysql_affected_rows($sql)>0) {
    //authenticated!
  }
?>
```

CONSEQUENCES

- if script does not make input analysis and validation, user can send

```
user = blah
pwd = ' OR user='blah'
```

- we get the query

```
SELECT * FROM users
WHERE user='blah' AND pwd='' OR user='blah'
```

- if at least one tuple does exist, attacker obtains authenticated access

OTHER (WORSE) CONSEQUENCES

- symbol ';' is exploited, it allows to concatenate commands

```
pwd = ' OR user='blah'; DROP TABLE users;
```

- or

```
pwd = ' OR user='blash'; INSERT INTO users
  (...) VALUES (...);
```

LINKS

- examples
  - http://www.owasp.org/index.php/SQL_Injection
  - http://www.unixwiz.net/techtips/sql-injection.html
- **Sqlninja**: example of tool for supporting attacks http://sqlninja.sourceforge.net/
  - it tries to use SQL injection on applications based on MS SQL Server
  - its goal is to obtain an interactive shell on the remote DB server
- **WebScarab**: example of tool for prevention http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
  - powerful, good prevention, even against other types of attack

PREVENTING SQL INJECTION

- *input validation*
  - client side
  - to be considered within the wider subject of software correctness and robustness
- *parameterized queries*
  - based on predefined query strings
- *use of stored procedures*
  - subroutines that are defined at server side, available to applications accessing the RDBMS
  - can validate input at server side

INPUT VALIDATION AT CLIENT SIDE

- use scripts, e.g., Javascript
- can be made weaker by the security settings of the browser
- in some cases, can be bypassed thru suitable change of the HTML source code

PARAMETERIZED QUERIES

- avoid the traditional dynamic query string, where pre-defined substrings have to be replaced by user defined text
- based on pre-defined query strings, where suitable parameters have to be inserted
- example: Java Prepared Statement

JAVA PREPARED STATEMENTS

- see Sun tutorial on JDBC (http://java.sun.com/docs/books/tutorial/jdbc/basics/index.html)
- technique based on Java class PreparedStatement
  - initially proposed for improving the speed of frequently executed queries
- when PreparedStatement is instantiated, an SQL query is built (and compiled): it may contain the symbol '?' to denote possible parameters necessary to query
- query structure is fixed

PRACTIC EXAMPLE

```
// define query schema
String selectStatement = "SELECT * FROM User WHERE
  userId = ? ";

// instantiate PreparedStatement object by means of
  purposed method of db connector (class Connection)
PreparedStatement prepStmt = con.prepareStatement
  (selectStatement);

// provide parameter thru setXXX
prepStmt.setString(1, userId); // 1 -> first
  parameter

// execute query
ResultSet rs = prepStmt.executeQuery();
```

VULNERABILITIES IN PREPARED STATEMENTS

- Java prepared statements, if not carefully packed, may be vulnerable to SQL injection
- example

```
String strUserName =
  request.getParameter("Txt_UserName");
PreparedStatement prepStmt =
con.prepareStatement("SELECT * FROM user
  WHERE userId = '+strUserName+'");
```

- a prepared statement is built, using a non-validated input parameter!

STORED PROCEDURES: WHAT AND WHY

- compiled procedures (subroutines) made available at server side to build/support batches operating on DB
- code is optimized, but DB server incurs higher processing costs
  - also improve code readability
- they help to limit SQL injection attacks
  - but they are not exempt from vulnerabilities

USE OF STORED PROCEDURES

- also known as *proc*, *sproc*, *StoPro* or *SP*, belong to *data dictionary*
- typical uses
  - data validation
  - access control mechanisms
  - centralization of logic that was initially contained inside the applications
- similar to the user-defined functions, but with different syntax
  - functions can appear everywhere in SQL strings, this is not true for stored procedure calls

DATA VALIDATION THRU SP

A few controls (partial list)

- format (e.g., digits or dates)
- types (e.g., if text has been inserted when digits are expected)
- range (check data that should belong to an admissible interval)
- mandatory data
- parity control
- orthography and grammar
- consistence M/F, S/P
- cross-system consistence (data on several systems; e.g., name + surname vs. surname + name)
- existence of referred files