Practical attack & defense

Lecture material 31.10-16.12.
Kimmo Saurén

SQL-injections

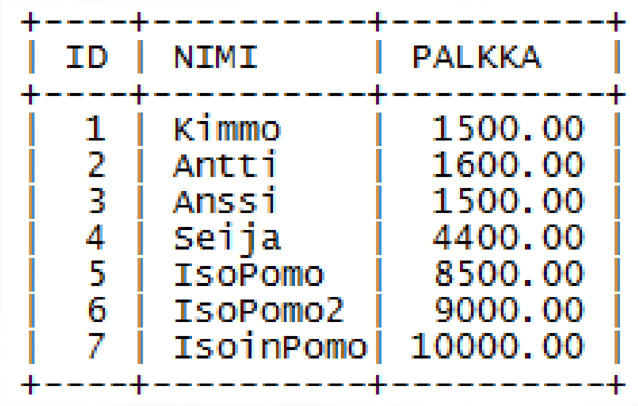
SQL

- Structured Query Language
- Standard query language developed by IBM
  - Makes a standardized interface to database
- In practice there exits three slightly different implementations
  - MySQL
  - MS IIS
  - Oracle
- Most used statements
  - SELECT, UPDATE, INSERT, DELETE
- Statements to control the structure of database
  - CREATE TABLE, CREATE VIEW

SQL

- Usage starts with selecting the database to work with
- *SHOW DATABASES;*

  USE companypeople;
- The data is in tables
  - arranged into columns and rows



```
+----+-----------+----------+
| ID | NIMI      | PALKKA   |
+----+-----------+----------+
|  1 | Kimmo     |  1500.00 |
|  2 | Antti     |  1600.00 |
|  3 | Anssi     |  1500.00 |
|  4 | Seija     |  4400.00 |
|  5 | IsoPomo   |  8500.00 |
|  6 | IsoPomo2  |  9000.00 |
|  7 | IsoinPomo| 10000.00 |
+----+-----------+----------+
```

- Data is accessed by using SELECT statement

  SELECT nimi, palkka FROM slaves;

  *SELECT * FROM slaves;*

SQL

- Results of query can be limited with WHERE clause

  http://www.victim.com/produts.asp?id=6+6

  *SELECT * FROM slaves WHERE palkka=2000;*

  *SELECT * FROM slaves WHERE palkka > 4000;*

  *SELECT * FROM slaves WHERE 1=1;*

- Where clause should be understood as a condition that has to be true with some rows of the database
- *SELECT * FROM slaves WHERE palkka > 2000 AND nimi='seija';*

  *SELECT * FROM slaves WHERE palkka > 2000 OR nimi='seija';*

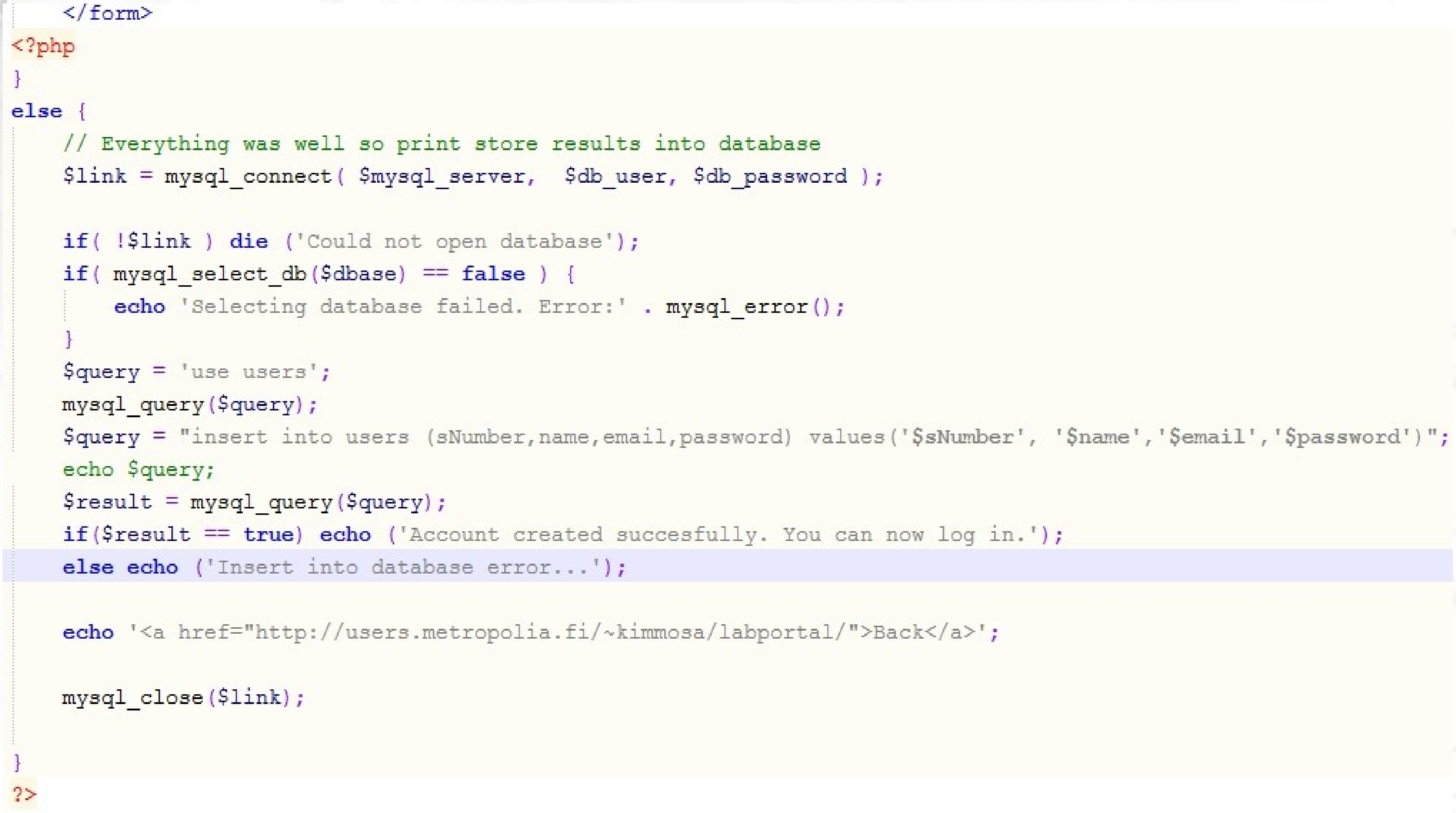
SQL

- Updating of database is done by UPDATE statement
- *UPDATE slaves SET palkka=9499 where nimi='kimmo';*
- One can add data by using INSERT INTO statement
- *INSERT INTO slaves (nimi, palkka) VALUES ('kaveri', 3000);*

- Removing data is done by DELETE statement
- *DELETE FROM slaves WHERE name='IsoPomo';*
- Databases or tables can be destroyed by using DROP TABLE / DROP DATABASE

PHP and SQL together

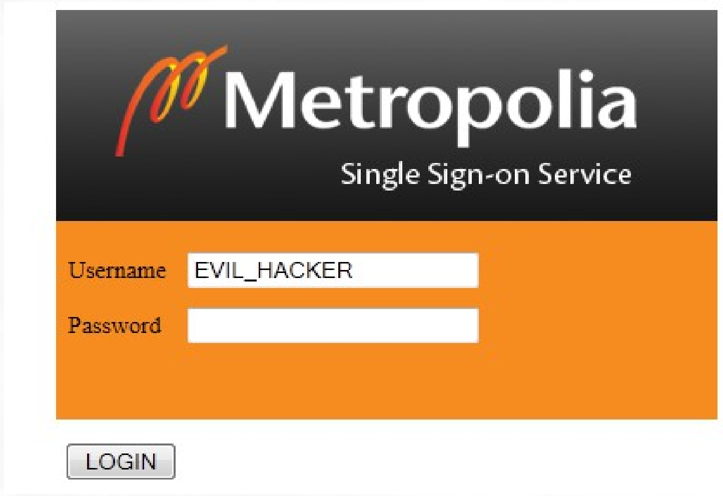
- SQL queries are made with PHP-language and html-page is constructed by using query results



```
    </form>
<?php
}
else {
    // Everything was well so print store results into database
    $link = mysql_connect( $mysql_server,  $db_user, $db_password );

    if( !$link ) die ('Could not open database');
    if( mysql_select_db($dbase) == false ) {
        echo 'Selecting database failed. Error:' . mysql_error();
    }
    $query = 'use users';
    mysql_query($query);
    $query = "insert into users (sNumber,name,email,password) values('$sNumber', '$name','$email','$password')";
    echo $query;
    $result = mysql_query($query);
    if($result == true) echo ('Account created succesfully. You can now log in.');
    else echo ('Insert into database error...');

    echo '<a href="http://users.metropolia.fi/~kimmosa/labportal/">Back</a>';

    mysql_close($link);

}
?>
```

SQL-injection

- SQL injection vulnerability is possible when user input is not properly validated and filtered



Username EVIL_HACKER

Password

- $input = $_POST['user'];

$query = ”SELECT * FROM users WHERE user='$input';”

- In the example above the user value is used without any sanitation
  - This makes SQL injection possible

SQL-injection

- In PHP SQL query is formed in the following way
- $query = ”SELECT * FROM users WHERE user='$input';”

  $\downarrow$

  SELECT * FROM users WHERE user='EVIL_HACKER';

- Now if user would give following input
- HACK' OR '1'='1 $\longrightarrow$

  SELECT * FROM users WHERE user=' HACK' OR '1'='1 ';
- In this case the meaning of WHERE clause is altered
  - Either user is HACK or one equal to one
- One equals to one is ofcourse always true so user would be logged on with the first user entry in database
  - Usually that is admin

SQL injection

- Easiest way to notice SQL injection is to put ' into a input field
  - If the field is vulnerable there will be SQL error because of the extra '
  - Next actions depend on how much information is available from the error
- Noticing SQL injection vulnerabilities are not that easy usually
  - SQL error notifications are not usually visible for users
- What can be done with vulnerability depends on SQL server configurations and permissions
  - If usage of certain SQL statements are not prohibited, it is possible to compromize whole server
  - Stealing all information is usually possible

SQL injection

- SQL injection vulnerability phases
  - Finding a vulnerability
  - Finding out the type and version of SQL server
  - Finding out the type of vulnerability
  - Resolving the structure of database
  - Stealing or changing information
  - Using the information
  - Taking control of server and installing backdoors

How to find SQL injection vulnerabilities

- All is based on giving input and analyzing responses
  - Non-normal responses might because of vulnerabilities
- GET request
  - One can change parameters directly from URL
- POST request
  - One can't change parameters directly
  - Requires a tool like Burp-proxy or Paros proxy
- Can be automated with tools like
  - Sqlninja
  - Sqlmap
  - Paros Proxy

How to find SQL injection vulnerabilities

- SQL injection is not limited just to POST/GET parameters but it is possible to find vulnerabilities elsewhere
  - cookies
  - Content headers
  - Browser type & name fields
- One should realize that all information that is relayed to server is a possible attack vector
  - Also values that are sent by server and are relayed back
- Real life SQL-injections

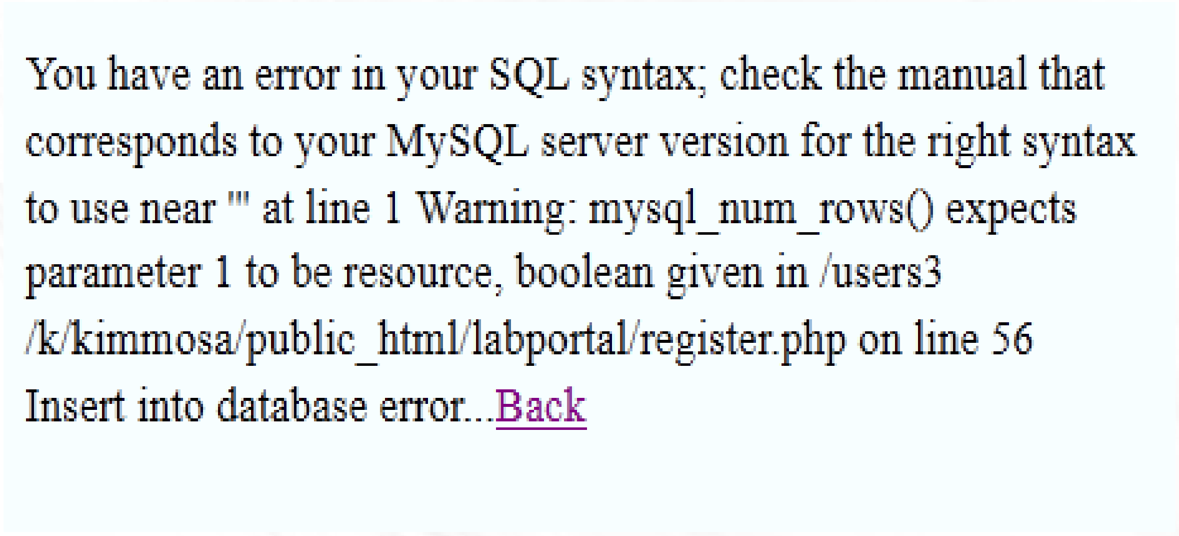
How to find SQL injection vulnerabilities

- Code inspections if source code is available
  - Code is inspected line by line and following the user input
  - Data sources and sinks should be identified
- Dangerous PHP coding habits

  *SELECT * FROM slaves WHERE nimi='$_POST[user]';*
- *Source code analysing can be automated somewhat*
  - *YASCA: Yet Another Source Code Analyzer*
  - *Pixy*
  - *AppCodeScan*

Normal SQL injection

- If SQL error notification is show, it is possible to gain information about structure of database
- In the case below application programmer has not filtered error messages



```
You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax
to use near ''' at line 1 Warning: mysql_num_rows() expects
parameter 1 to be resource, boolean given in /users3
/k/kimmosa/public_html/labportal/register.php on line 56
Insert into database error...Back
```

Normal SQL injection

- Printing out for example the version of server is possible by using error messages

http://www.victim.com/products.asp?id=@@version

Normal SQL injection

- Finding out the structure with using UNION statement
  - Using UNION it is possible to combine two SELECT statements into one result output
  - Both SELECT statements have to have equal amounts of columns
  - The number of columns can be guessed by using ”null” instead with real names
  - Error message is given if the number of ”null”s is not correct

SELECT * FROM products WHERE id=12 UNION SELECT null, null, null;

.http://www.victim.com/list.php?id=12+union+select+null,null,null

Normal SQL injection

- Printing out information using UNION statement
  - Column types of two SELECT statements have to be same
  - For example if it is wanted to print out string type information, one would have to have a string type column in vulnerable SQL query

SELECT * FROM products WHERE id=12 UNION SELECT system_user, null, null;

.http://www.victim.com/list.php?id=12+union+select+system_user,null,null

Normal SQL injection

- Finding out the structure using ORDER BY clause
  - Using ORDER BY clause is better than using UNION clause since it makes only one unsuccesful SQL query
  - The less unsuccesful SQL queries, the more quieter the attack method is

```
SELECT * FROM products WHERE id=12 ORDER BY 3;
.http://www.victim.com/list.php?id=12+order+by+3
```

Blind SQL injection

- Blind injection is a type a vulnerabilty where there is no error messages to be used
  - The application programmer has filtered database errors as they should
- Blind injection type is hard to notice
  - It is based on noticing a difference between succesful and unsuccesful queries
  - The difference can be really minimal, so the results has to be analyzed very carefully
- Finding out blind injection vulnerability

*SELECT * FROM slaves WHERE palkka > 2000 AND 1=1;*

*SELECT * FROM slaves WHERE palkka > 2000 AND 1=0;*

Blind SQL-injections

- With making true/false queries it is possible to find out the differencies between true and false responce
  - Blind injection vulnerability is found if there is a difference
  - A application can be still vulnerable even tough there are no differencies in responcies
    - Differencies in responcies can be found out with timed query

http://www.victim.com/produts.asp?id=12;if+(1=1)+WAITFOR+ DELAY+'0:0:5'--

- How to find out the database structure in blind injection case
  - There is no easy way since no error messages are received
  - Column names has to be guessed or use brute force methods
  - There are tools available for using brute force methods

Blind SQL-injections

- Parameter division technique
  - It is possible to find out blind injection if parameter division gives the same result

http://www.victim.com/produts.asp?id=12

http://www.victim.com/products.asp?id=6%2B6

- Parameter division technique is not limited to numbers but it is possible to divide strings also
  - If web application is made correctly parameter division should not work
  - If not the splitted parameter is relayed to SQL server which will make the calculation (6+6) and return correct results

Common blind injection cases

- Malformed SQL query gives a error page and correct SQL query gives a page which is controllable
  - For example following a normal link lists items, but modifying URL gives you a error page
  - In this case SQL injection vulnerability can be verified with timed SQL injection test
- Malformed SQL query gives you a error page and correct SQL guery a page that is not cotrollable
  - For example if page is using UPDATE/INSERT SQL statement which inserts data into table and does not show any results
- Malformed SQL query gives no error page and no other page at all
  - Vulnerability is verified with timed SQL injection

Blind SQL-injection

- Vulnerability can be exploited by using differencies in responces
  - Adding a not true component into working SQL query

    ```
    SELECT * FROM slaves WHERE palkka=2000;            // True
    SELECT * FROM slaves WHERE palkka=2000 AND 1=2; // Not true
    ```

  - One can obtain information by knowing the difference

```
SELECT * FROM slaves WHERE palkka=2000 AND
  SUBSTRING(SYSTEM_USER, 1, 1)='a';
SELECT * FROM slaves WHERE palkka=2000 AND
  SUBSTRING(SYSTEM_USER, 1, 1)='b';
SELECT * FROM slaves WHERE palkka=2000 AND
  SUBSTRING(SYSTEM_USER, 1, 1)='c';
```

- Tools can be found to automate this

Defending from SQL injections

- Basic methods
  - Good programming guidlines
  - Education about SQL injections and consequeces of it
  - Automated tools to test applications
  - Code reviews
- Technical methods
  - Unusual database and table names
  - Honeypot databases and tables
    - For example password table which send a email to administrator if ever accessed
  - Storing passwords in coded form (SHA-1)
  - All important information should be in coded form
  - Using multiple databases and servers

Defending from SQL injections

- Parametrized SQL queries
  - SQL injection is not possible

    *$sql = SELECT * FROM users WHERE user=? AND password=?;*

    $con->prepare($sql);

    $con->bind_param(”ss”, $username, $password);
  - *$con->execute();*
- Filtering user input
  - White listing: allows certain ascii characters and disallows rest
  - Black listing: disallows certain ascii characteds and allows rest
  - Filtering can be done by using regular expressions

Defending from SQL injections

- Encoding
  - Protecting SQL queries by encoding special characters before sending them to SQL server
  - For example PHP mysql_real_escape_string() adds \ before each special character so it is not interpreted as a part of control