Implementation PDO Parameterized Query to Prevent SQL Injection

Maksy Sendiang
Information Technology Dept
Manado State Polytechnic
North Sulawesi - Indonesia

Ottopianus Mellolo
Information Technology Dept
Manado State Polytechnic
North Sulawesi - Indonesia

Maureen Langie
Electrical Engineering Dept
Manado State Polytechnic
North Sulawesi - Indonesia

ABSTRACT
SQL injection is one of threat to the application connected to the database. By implementing SQL injection attacker can gain full access to the application or database so that it can remove even significant data irresponsibly. Applications that do not validate the user's input appropriately make them vulnerable against SQL injection. Various methods have been developed to prevent SQL injection each with advantages and disadvantages. Implementation of PDO Parameterized Query properly can prevent SQL injection. . PDO not only provides a method to facilitate the implementation of parameterized queries but also makes the code is portable because the PDO can be used on multiple databases. This paper describes the results of research on the use of PDO Parameterized Query on scheduling application. By using PDO Parameterized Query on this application, making it is not vulnerable to attack that caused by SQL injection.

General Terms
Internet security, web technology, object oriented programming

Keywords
PDO, parameterized query, SQL injection

1. INTRODUCTION
Data and information has become a very important asset in both government and private organizations nowadays. Every day large amounts of data stored using hardware or software . Usage and rapid progress of the Internet infrastructure , has spurred the increasing amount of data stored in a database lately . Increasing the number of users and the heavy reliance on digital information is one barometer of the importance of securing data or information [1].

Web applications have become the interface is widely used in presenting data and information. Most web applications using multitier design that consists of a presentation tier (front end) , application tier (middle tier) and a data tier (backend) [2]. Presentation tier is the leading layer that presents information regarding to services presented by the web application. Application layer is a layer that implements software functionality by performing the process in detail. . The data tier is a layer that stores data and provides a response to a request from the application tier and consists of a database server. These tiers shaping the architecture of client – server developed as separate modules that are generally known as a user interface module, functional process logic module and data storage module.

The software includes web applications vulnerable to security threats. The web application with a database that stores important information such as financial information , health information , personal information to be very sensitive and is one of the targets of the Structured Query Language (SQL) injection. According to the Open Web Application Security Project (OWASP) [3] SQL Injection ranks first in the list of web application security threats in 2013. In terms of avoiding from the threat of SQL Injection then access to the database should be monitored by setting a parameterized query using PHP Data object (PDO)

2. LITERATURE REVIEW
2.1 Definition of SQL Injection
SQL injection is a type of security exploit in which the attacker adds Structured Query Language (SQL) code to a web form input box, to access the gain resources or make changes to data [4]. SQL injection is done by attacker with transmit a particular code into a SQL query so a SQL code is formed. In this way , the attacker can gain access to the databases that store important and crucial data. This matter is very risky because it can cause data loss or misuse of data by parties who are not responsible.

According to [5] SQL injection can be implemented in several ways including by tautology attacks , union attacks , logically incorrect query attacks and piggy back attacks. Tautology attacks carried out by inserting a conditional statement into the query to create the conditions are always right. Union attacks carried out by adding the UNION keyword in the query to perform operations is undesirable in a database . Piggy back attacks carried out by adding a query that has been injected into the original query.

2.2 SQL Injection Prevention Methods
SQL injection is implemented by tautology can be prevented by input validation e.g to check the type , size , format and range of data entered and the data entered should not be to pass the authentication process .

Setting the field of the length to limit the number of characters that can be used and to encrypt information can prevent SQL injection is implemented by way of UNION attacks..

2.3 PHP Data Object
PDO extension has become one of the trends in developing dynamic web applications and connect to the database . PDO is the PHP5 extensions written in C / C ++ and has several advantages including the system supports a number of databases supported by PHP , faster because it is written with a compiled language , and easy installation . In short PDO needed when we needed a portable application that supports a number of database systems and faster execution .

PDO provides database abstraction layer that can use the same functions to execute SQL commands on any database [6] . The main reason to use PDO is security and flexibility when connected to the system database. Through the use of

prepared statements (utility not in php_mysql *) then the PDO can prevent SQL Injection [7] .

3. .METHODOLOGY

This study uses research and development method that includes four phases: analysis, design, implementation and testing. Implementation PDO parameterized query to prevent SQL Injection in this paper is applied to the scheduling application for vocational high school in North Sulawesi Province of Indonesia. This object -oriented applications in development using Rational Unified Process (RUP) . This method is used because the time needed in application development is relatively short and this application will undergo repairs during the development process

Rational Unified Process (RUP) is a software development approach that is done iteratively, focusing on architecture (architecture- centric) and is directed by use cases. RUP is a software engineering process of good defining and structuring . RUP provides a good structure for defining workflow software project life [8] .

RUP has four stages or phases that can be done iteratively. In this methodology, there are four stages of software development, eg:

1. Inception is a stage model the business processes required and defines the need for the system to be created
2. Elaboration is more focused on planning the system architecture. This stage can also be made to determine whether the desired system architecture can be made or not. This stage also gives emphasis on the analysis of the system design and system implementation and expected results of this phase is to fulfill the Lifecycle Architecture Milestone
3. Construction, this stage is more focused on the development of a component or system features
4. Transition, this stage is the deployment or installation of the system in order to be understood by the user. Activities at this stage includes user training, maintenance and testing of the system to meet user expectations

4. RESULT AND DISCUSSION

This paper describes how to implement the PDO Parameterized Query to prevent SQL injection . According to [4] vulnerabilities on web applications that can lead to SQL injection is as follows :

1. SQL Injection is based on a 1 = 1 is always true. Look at SQL query in Figure 1 below

```
$sql = "SELECT * FROM penggunawhere alias= '$nama'
and password = '$pass';
```

Fig 1: Query Example

SQL code above is used to select user based on name, passwords and levels of the following form (see Figure 2) :

Fig 2: Login Form

If no validation, the user can enter any input, including input that can exploit security as in the following query (Figure 3):

```
$sql = "SELECT * FROM penggunawhere alias= '' OR1=1
and password = '' OR1=1;
```

Fig 3: SQL Query with 1 = 1

2. SQL Injection based on batched SQL Statements. Most databases that supports batched SQL statements separated by a semicolon, as shown in Figure 4 below

```
$sql = "SELECT * FROM pengguna; DROPTABLE
pengguna
```

Fig 4: Batched SQL Statements

SQL code above will display the user table and then delete the table .

3. SQL injection based on commenting password. For example user adds username and password like picture below;

```
Username = 'admin'---
Password = *****
$sql = "SELECT * FROM pengguna where alias
= 'admin'—and password = *****
```

Fig 5: SQL injection with commenting password

If the query in Figure 5 above is executed , the query will check alias and password will not be checked because the '--' symbol is used as a comment on SQL .

PDO Parameterized Queries can prevent SQL Injection . With a parameterized query the database will be able to distinguish between SQL commands and data entered by the user . If the SQL commands entered by the attacker , parameterized query would treat it as an input that is not trusted (untrusted input) and SQL commands that are SQL injection will never be executed . The following example shows the syntax of PHP snippet that contains loopholes SQL Injection

```
<?php
if (isset($_GET['id'])){
 $id = $_GET['id'];
 $mysqli = new mysqli('localhost', 'dbuser', 'dbpasswd', 'hibah');
 if ($mysqli->connect_errno) {
```

```
printf("Connect failed: %s\n", $mysqli->connect_error);
exit();
}
/* SQL query contains SQL injection */
$sql = "SELECT username FROM users WHERE id = $id";
if ($result = $mysqli->query($sql)) {
while($obj = $result->fetch_object()){
print($obj->username); } }
```

In order to prevent SQL Injection then Parameterized Query PDQ implemented as in the following code snippet

```
try{ //
    $dbh = new PDO('mysql:host=localhost; dbname =hibah', 'dbuser', 'dbpasswd');
    /**
     * Use PDO::ERRMODE_EXCEPTION, to catch errors and write it to a log file for the future
     */
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    /**
     * Before executing SQL statements,we need to set binding parameters. The following syntax fix the SQL injection vulnerability
     */
    $q = "SELECT username FROM users WHERE id = :id";
    $sth = $dbh->prepare($q);
    $sth->bindParam(':id', $id);
    $sth->execute();
indexed by column name
    $sth->setFetchMode(PDO::FETCH_ASSOC);
    $result = $sth->fetchColumn();
    print( htmlentities($result) );
        $dbh = null;
   }
   catch(PDOException $e)
{
    error_log('PDOException - ' . $e->getMessage(), 0);
     http_response_code(500);
    die('Error establishing connection with database');
   }
  } else{
   http_response_code(400);
   die('Error processing bad or malformed request');
    }
  }
```

Scheduling application for vocational high school in North Sulawesi Province of Indonesia applied PDO Parameterized Query to create applications capable of counteracting SQL Injection attacks. There are some reason using PDO instead of mysql_* in this application, as shown in table 1.

Table 1. Comparison between mysql_* and PDO

No	Mysql_*	PDO
1	It doesn't support modern SQL database concept such as prepared statements, stored procedures, transactions, etc	It supports modern SQL database
2	It concatenates escaped strings into SQL	Using bind parameters which is an easier and cleaner way of securing queries
3	It lacks consistent error handling or no handling at all	Using exception mode in error handling
4	It is not being maintained (security vulnerabilities are not getting fixed)	It is being maintain and much more secure

Prevention of SQL injection starts from use the following syntax to create a connection to a mysql database:

```
<?php
  class CekLogin
  {
    var $dbh;
    function __construct()
    {
        $hostname = 'localhost';
        $username = 'maksy';
        $password = 'makk7423';
        try
        {
          $dbh = new PDO("mysql:host=$hostname ;
          dbname=hibah", $username, $password);
          $dbh->setAttribute(PDO : : ATTR_ERRMODE,
          PDO : : ERRMODE_EXCEPTION);
          $dbh->setAttribute(PDO : :
          ATTR_EMULATE_PREPARES, false);
          $this->dbh = $dbh;
        }
        catch(PDOException $e)
        {
          echo $e->getMessage();
        }
    }
```

Fig 6 : PDO – mysql Connection

Compared with traditional syntax to connect php mysql, the above syntax is much more secure against SQL injection. Traditional syntax relies on die() to handle errors but in fact php application cannot handle die(), as it will just end the script abruptly and then echo the error to the screen and allow nasty hackers discover the schema and do the SQL injection. On the other hand, PDO has three error handling modes e.g :

1. *PDO::ERRMODE_SILENT* acts like mysql_* where it will check each result and then look at *$dbh->errorInfo ( );* to get the error details.
2. *PDO::ERRMODE_SILENT* throws PHP warnings
3. *PDO::ERRMODE_EXCEPTION* throws PDOException. This mode is used in this scheduling application, it acts very much like *die (mysql_error( );* when it isn't caught but unlike *die( )* the PDOException can be caught and handled gracefully

In this scheduling application, fetching data based on key supplied by a form. The user input is automatically quoted, so there is no risk of a SQL injection attack (see fig 7).

```
try
{
  $sql = "SELECT * FROM pengguna where nama = ? and
password = ? and level = ?";
$count = $this->getDatabase()->query($sql)->rowCount();
if($count > 0 )
{
   try
   {
      ……
```

Fig 7 : Fetching data

PDO Parameterized Query is used to perform filtering on any data entered by the user and it proved able to cope SQL Injection, as shown in one of the following function below

```
function addTeacher($nip,$nama,$alamat,$telp)
{   $log = new CekLogin();
    try
    {
        $sth = $log->getDatabase()->prepare ("INSERT
        INTO teacher (nip,nama,alamat,telp)VALUES
        (?,?,?,?)");
        # use bindValue() to bind data values
        $sth->bindValue (1,$nip);
        $sth->bindValue (2,$nama);
        $sth->bindValue (3,$alamat);
        $sth->bindValue (4,$telp);
        $sth->execute ();
      //echo "Data success to be added";
    }
        catch(PDOException $e)
        { echo $e->getMessage();}}
```

In the above code SQL statement that is passed to the method prepare () will be parsed and compiled by the database server. By declaring parameters in the form of "?" will be signaled to the database server where the screening process will be conducted. When the method execute () is called, the statements contained in the method prepare () will be aggregated with the value of the parameter that has been determined. It should be emphasized that the values of the parameters, coupled with the SQL statement that was compiled not by SQL as a string. SQL injection works by adding malicious code as a parameter to the SQL statement. Because the SQL statement and parameters is executed separately, it will reduce the impact of things that are not desirable. Each parameter is sent through methods prepare () will be treated as a string.

In addition, by using a parameterized query through the application of methods prepare ( ) can improve system performance because the SQL statement is compiled only once and used repeatedly with different parameter values.

All PDO parameterized query techniques mentioned above apply to this scheduling application that consists of several modules. The result of applying PDO parameterized query on login module can be seen in the following table :

Table 2. The Results of the implementation of the PDO on the login module

Username	Password	Level	Status
'' OR 1=1	'' OR 1=1	'' OR 1=1	Prevented
'admin'--	Random	student	Prevented
admin	SQL statement	teacher	Prevented
admin;	pass	teacher	Prevented
admin	admin	teacher	Login granted
admin	wrong password	teacher	Invalid entry

5. CONCLUSION

Use of PDO Parameterized Queries can prevent SQL Injection and also can improve system performance through the use of methods prepare ( ) and the application of object-oriented php. Application of PDO parameterized query in the development of web-based applications will soon replace the mysql_* function. Therefore, the web application developers should have the knowledge to use PDO parameterized query. The features of PDO continues to grow because it supported by the PHP developers worldwide. Application of PDO Parameterized Query on scheduling applications in North Sulawesi Province - Indonesia makes this application is not vulnerable to SQL Injection threat .

6. REFERENCES

[1] Yash Tiwari, Mallika Tiwari, "A study of SQL of injection techniques and their prevention methods", International Journal of Computer Applications (0975-8887), vol 114, no. 17, March 2015.

[2] Bojken Shehu, Aleksander Xhuvani, "A literature review and comparative analyses on SQL injection : vulnerabilities, attcks and their preventation and detection techniques", IJCSI International Journal of Computer Science Issues, vol 11, issue 4, no. 1, July 2014

[3] The Open Web Application Security Project, "OWASP TOP Project", https://www.owasp.org/SQL_Injection

[4] Bharti Nagpal, Naresh Chauhan, Nanhay Singh, "A viable solution to prevent SQL injection attack using SQL injection", i-manager's Journal on Computer Science, vol.3, no.3, September – November 2015

[5] Yogesh Bansal, Jin Park, "Multi-hashing for protecting web applications from SQL injection attacks", International Journal of Computer and Communication Engineering,vol.4,no.3, May 2015

[6] Mandalika G, "Developing MySQL database application with PHP Part 3 : using the PDO extension with MySQL driver", Maret 2009. URL : http://www.oracle.com /technetwork/systems/articles/mysql-php3-140148.html, diakses tanggal 28 Desember 2015.

[7] Utami E, Raharjo S,"Database Security Model in the Academic Information System", International Journal of Security and Its Applications. 8:170. 2014

[8] Chen Q, "Compare and study about owing to the three kinds important softwaresdevelop process", Proceeding of the International Conference on Education Technology and Economic Management (ICETEM). 450-451. 2015