ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

IT4552 – Web programming

Chapter 6.2. MySQL & PHP Advanced

1

Content

2

Database queries with PHP
(the wrong way)

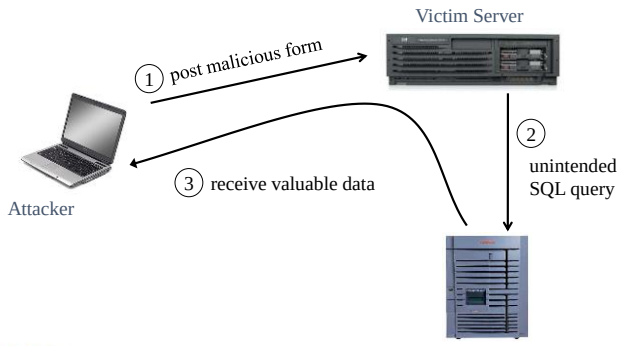
- Sample PHP

```
$recipient = $_POST['recipient'];
$sql = "SELECT PersonID  FROM People WHERE
 Username='$recipient' ";
$rs = $db->executeQuery($sql);
```

- Problem:
  - Untrusted user input 'recipient' is embedded directly into SQL command

3

Basic picture: SQL Injection



4

CardSystems Attack

- CardSystems
  - credit card payment processing company
  - SQL injection attack in June 2005
  - put out of business
- The Attack
  - 263,000 credit card #s stolen from database
  - credit card #s stored unencrypted
  - 43 million credit card #s exposed

5

April 2008 SQL Vulnerabilities

Brian Krebs on Computer Security

About This Blog | Archives | XML RSS Feed (What's RSS?)

Hundreds of Thousands of Microsoft Web Servers Hacked

Hundreds of thousands of Web sites - including several at the **United Nations** and in the U.K. government -- have been hacked recently and seeded with code that tries to exploit security flaws in **Microsoft Windows** to install malicious software on visitors' machines.

The attackers appear to be breaking into the sites with the help of a security vulnerability in Microsoft's Internet Information Services (IIS) Web servers. In an alert issued last week, Microsoft said it was investigating reports of an unpatched flaw in IIS servers, but at the time it noted that it wasn't aware of anyone trying to exploit that particular weakness.

Update, April 29, 11:28 a.m. ET: In a post to one of its blogs, Microsoft says this attack was *not* the fault of a flaw in IIS: "...our investigation has shown that there are no new or unknown vulnerabilities being exploited. This wave is not a result of a vulnerability in Internet Information Services or Microsoft SQL Server. We have also determined that these attacks are in no way related to Microsoft Security Advisory (951306). The attacks are facilitated by SQL injection exploits and are not issues related to IIS 6.0, ASP, ASP.Net or Microsoft SQL technologies. SQL injection attacks enable malicious users to execute commands in an application's database. To protect against SQL injection attacks the developer of the Web site or application must use industry best practices outlined here. Our counterparts over on the IIS blog have written a post with a wealth of information for web developers and IT Professionals can take to minimize their exposure to these types of attacks by minimizing the attack surface area in their code and server configurations."

Shadowserver.org has a nice writeup with a great deal more information about the mechanics behind this attack, as does the SANS Internet Storm Center.

6

Main steps in this attack

- Use Google to find sites using a particular ASP style vulnerable to SQL injection
- Use SQL injection on these sites to modify the page to include a link to a Chinese site nihaor1.com

  (Don't visit that site yourself!)
- The site (nihaorr1.com) serves Javascript that exploits vulnerabilities in IE, RealPlayer, QQ Instant Messenger

Steps (1) and (2) are automated in a tool that can be configured to inject whatever you like into vulnerable sites

7

Example: buggy login page (ASP)

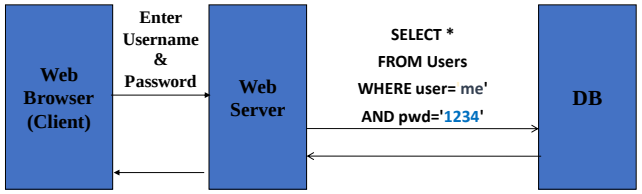
```
set ok = execute( "SELECT * FROM Users
     WHERE user=' " & form("user") & " '
     AND   pwd=' " & form("pwd") & " '" );

if not ok.EOF
     login success
else  fail;
```

Is this exploitable?

8

Normal Query



9

Bad input

- Suppose user = " `'or 1=1 --` " (URL encoded)
- Then scripts does:

```
ok = execute( SELECT …
        WHERE user= ' ' or 1=1  -- … )
```

  - The "`--`" causes rest of line to be ignored.
  - Now ok.EOF is always false and login succeeds.
- The bad news: easy login to many sites this way.

10

Even worse

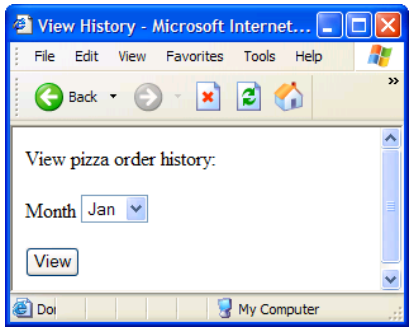
- Suppose user =
  " ' ; DROP TABLE Users -- "
- Then script does:

```
ok = execute( SELECT …
  WHERE user= ' ' ; DROP TABLE Users  …  )
```

- Deletes user table
  - Similarly: attacker can add users, reset pwds, etc.

11

Getting private info



13

Getting private info

SQL Query

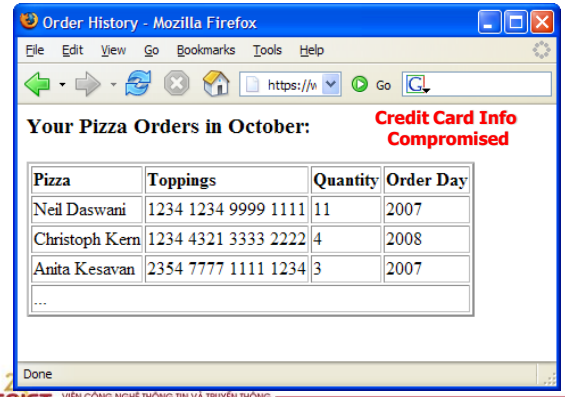
```
"SELECT pizza, toppings, quantity, date
  FROM orders
  WHERE userid=" . $userid .
"AND order_month=" . _GET['month']
```

What if:

```
month = "
    0 AND 1=0
    UNION SELECT  name,  CC_num,  exp_mon,  exp_year
    FROM  creditcards  "
```


14

Results



15

Preventing SQL Injection

- Never build SQL commands yourself!
  - Using mysql_real_escape_string(): Escapes special characters in a string for use in a SQL statement
  - Use parameterized/prepared SQL
  - Use ORM (Object Relational Mapper) framework.

16

Parameterized/prepared SQL

- Builds SQL queries by properly escaping args: ' → \'
- Example: Parameterized SQL: (ASP.NET 1.1)
  - Ensures SQL arguments are properly escaped.

```
SqlCommand cmd = new SqlCommand(
    "SELECT * FROM UserTable WHERE
    username = @User AND
    password = @Pwd", dbConnection);
cmd.Parameters.Add("@User", Request["user"] );
cmd.Parameters.Add("@Pwd", Request["pwd"] );
cmd.ExecuteReader();
```


17

Parameterized/prepared SQL in PHP – using mysqli

```
<?php
 $mysqli = new mysqli("localhost", "me", "mypass", "world");
 if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
 }

 $city = "Amersfoort";
 $stmt =  $mysqli->stmt_init();
 if ($stmt->prepare("SELECT District FROM City WHERE Name=?")){
    $stmt->bind_param("s", $city);
    $stmt->execute();
    $stmt->bind_result($district);
    $stmt->fetch();
    printf("%s is in district %s\n", $city, $district);
    $stmt->close();
 }
 $mysqli->close();
?>
```


18

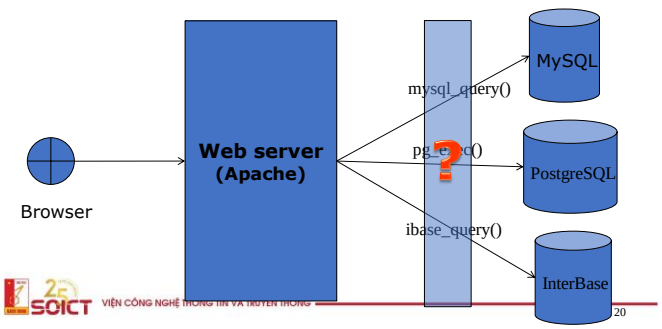
Content

1. SQL Injection
2. PEAR DB Basics
3. Advanced Database Techniques

19

2.1. Different database engines – Issue in PHP

- no general-purpose database access interface
- separate sets of functions for each database system



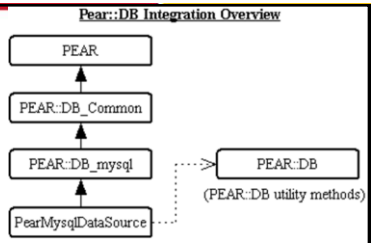
20

2.1. Different database engines - Solutions

- Provide a DB common mechanism to connect and manipulate to any database
- Some popular modules/libraries/extensions/APIs:
  - PDO (PHP Data Object)
    - provides a *data-access* abstraction layer
  - PEAR (the PHP Extension and Add-on Repository)
    - provides an abstract interface that hides database-specific details and thus is the same for all databases supported by PEAR DB
  - PHP Database ODBC
    - an API that allows you to connect to a data source
    - ODBC connection must be available

21

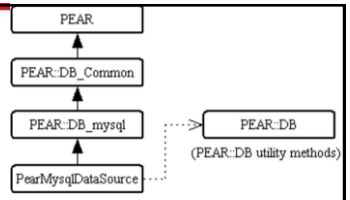
2.2. PEAR DB Overview
(From PHP4 and up)



- Two-level architecture:
  - **The top level:** provides **an abstract interface** that hides database-specific details and thus is the same for all databases supported by PEAR DB.
  - **The lower level:** consists of **individual drivers**, each driver supports a particular database engine and translates between the abstract interface seen by script writers and the database-specific interface required by the engine.

22

2.2. PEAR DB Overview



- 2 files used for all database engines
  - *DB.php*: Implements the DB class that creates database connection objects, and also contains some utility routines.
  - *DB/common.php* implements the DB_common class that forms the basis for database access.
- 1 file chosen on an engine-specific basis:
  - *DB/driver.php (E.g. DB/mysql.php):* Contains the driver for the database you're using. It implements DB_*driver* class that inherits DB_common class

23

2.3. Writing PEAR DB Scripts - Steps

- Reference the *DB.php* file to gain access to the PEAR DB module.
- Connect to the MySQL server by calling connect() to obtain a connection object.
- Use the connection object to issue SQL statements and obtain result objects
- Use the result objects to retrieve information returned by the statements.
- Disconnect from the server when the connection object is no longer needed.

24

2.3.1. Referencing the PEAR DB Source

- Before using any PEAR DB calls, your script must pull in the *DB.php* file

```
require_once "DB.php";
```


25

2.3.2. Connecting to the MySQL Server

- DSN (Data Source Name)
  - Contains connection parameters
  - URL-style includes the database driver, hostname, user name and password for your MySQL account, and the database name.
  - Typical syntax:

```
mysqli://user_name:password@host_name/db_name
```

  - E.g.:

```
$dsn = "mysqli://testuser:testpass@localhost/test";
$conn = DB::connect ($dsn);
if (DB::isError ($conn))
  die ("Cannot connect: ".$conn->getMessage()."\n");
```


Specifying connections parameters in a separate file

- Create a file *testdb_params.php*

```
<?php
  # parameters for connecting to the "test" database
  $driver = "mysqli";
  $user = "testuser"; $password = "testpass";
  $host = "localhost"; $db = "test";
  # DSN constructed from parameters
  $dsn = "$driver://$user:$password@$host/$db";
?>
```

- Include the file into your main script and use the $dsn variable

```
require_once "testdb_params.php";
$conn = DB::connect ($dsn);
if (DB::isError ($conn))
```


2.3.3. Issuing statements

- `$stmt = "some SQL statement";`
- `$result = $conn->query ($stmt);`
  - If an error occurs, `DB::isError($result)` will be true.
  - If the statement is INSERT or UPDATE, `$result` will be `DB_OK` for success.
  - If the statement is SELECT, `$result` is a result set object.

2.3.4. Retrieving result information

- Statements That Return **No** Result Set
  - Using `$conn->affectedRows()` to get no of rows the statement changed.
- Statements That Return **a** Result Set
  - Using `$result->fetchRow()` to get a row from result set. Result is an array including all cells in that row.
  - Using index to retrieve an element (cell) of the array of a specific row.
  - Using `$result->free()` to dispose `$result`
  - Using `$result->tableInfo()` to get detailed information on the type and flags of fields
    - `$info = $result->tableInfo();`

Issuing Statements That Return No Result Set

```
CREATE TABLE animal (
          name CHAR(40),
          category CHAR(40))
$result = $conn->query(
          "INSERT INTO animal (name, category)
               VALUES ('snake', 'reptile'),
                      ('frog', 'amphibian'),
                      ('tuna', 'fish'),
                      ('racoon', 'mammal')");
  if (DB::isError ($result))
      die ("INSERT failed: ".$result->getMessage());
  printf("\nNumber of rows inserted: %d\n",
                      $conn->affectedRows());
```


30

Issuing Statements That Return a Result Set

```
$result = $conn->query (
     "SELECT name, category FROM animal");
if (DB::isError ($result))
  die("SELECT failed: ".$result->getMessage());

printf ("Result set contains %d rows and %d columns\n",
         $result->numRows(), $result->numCols());
while ($row = $result->fetchRow())
  printf ("%s, %s\n", $row[0], $row[1]);
$result->free();
```


31

Issuing Statements That Return a Result Set – Other ways

- Optional argument for `fetchRow()` indicating what type of value to return
  - `DB_FETCHMODE_ORDERED` : refer to array elements by numeric indices beginning at 0.
  - `DB_FETCHMODE_ASSOC`: refer to array elements by column name
  - `DB_FETCHMODE_OBJECT`: access column values as object properties
- Setting fetching mode only one time:
  - `$conn->setFetchMode(DB_FETCHMODE_ASSOC);`

32

Example

```
while ($row = $result->fetchRow (DB_FETCHMODE_ASSOC))
   printf ("%s, %s\n",$row["name"],$row["category"]);
while ($obj = $result->fetchRow (DB_FETCHMODE_OBJECT))
   printf ("%s, %s\n", $obj->name, $obj->category);
$conn->setFetchMode (DB_FETCHMODE_ASSOC);
$result = $conn->query ($stmt1);
while ($row = $result->fetchRow ()) ...
...
$result = $conn->query ($stmt2);
while ($row = $result->fetchRow ()) ...
...
```


33

2.3.5. Disconnecting from the Server

- Close the connection when you're done using the connection:
  - `$conn->disconnect ();`

34

Content

1. SQL Injection
2. PEAR DB Basics
3. Advanced Database Techniques

35

3.1. Placeholders

- PEAR DB can build a query by inserting values into a template
- Syntax:
  - `$result = $conn->query(SQL, values);`
- E.g.

```
...
$books = array(array('Foundation', 1951),
               array('Second Foundation', 1953),
               array('Foundation and Empire', 1952));
foreach ($books as $book) {
      $conn->query('INSERT INTO books (title,pub_year)
                    VALUES (?,?)', $book);
}
```


36

3.1. Placeholders (2)

- Three characters as placeholder values
  - ?: A string or number, which will be quoted if necessary (recommended)
  - |: A string or number, which will never be quoted
  - &: Requires an existing filename, the contents of which will be included in the statement (e.g., for storing an image file in a BLOB field)

37

3.2. Prepare/Execute

- Using the **`prepare()`**, **`execute()`**, and **`executeMultiple()`** methods
  - **`$compiled = $db->prepare(SQL);`**
    (SQL using placeholders)
  - **`$response = $db->execute(compiled, value);`**
  - **`$responses = $db->executeMultiple(compileds, values);`**
    (takes a two-dimensional array of values)

38

Example - Prepare/Execute

```
$books = array(array('Foundation', 1951),
          array('Second Foundation', 1953),
          array('Foundation and Empire', 1952));
$compiled = $db->prepare('INSERT INTO
             books (title,pub_year) VALUES (?,?)');
foreach ($books as $book) {
    $db->execute($compiled, $book);
}
```

```
$books = array(array('Foundation', 1951),
                array('Second Foundation', 1953),
                array('Foundation and Empire', 1952));
$compiled = $db->prepare('INSERT INTO
                books (title,pub_year) VALUES (?,?)');
$db->executeMultiple($compiled, $books);
```


39

3.3. Sequences

- PEAR DB sequences are an alternative to database-specific ID assignment (for instance, MySQL's AUTO_INCREMENT).
- Create/drop a sequence
  - **`$res = $db->createSequence(sequence);`**
  - **`$res = $db->dropSequence(sequence);`**
- The nextID( ) method returns the next ID for the given sequence:
  - **`$id = $db->nextID(sequence);`**

40

3.3. Sequences (2) - Example

```
$res = $db->createSequence('books');
if (DB::isError ($result))
   die("SELECT failed: ".$result->getMessage());

$books = array(array('Foundation', 1951),
          array('Second Foundation', 1953),
          array('Foundation and Empire', 1952));
foreach ($books as $book) {
   $id = $db->nextID('books');
   array_splice($book, 0, 0, $id);
   $db->query('INSERT INTO
                books(bookid,title,pub_year)
              VALUES (?,?,?)', $book);
}
```


41

3.4. Shortcuts

- PEAR DB provides a number of methods that perform a query and fetch the results in one step, allowing placeholders
  - **`getOne(SQL [,values])`**: fetches the first column of the first row of data
  - **`getRow(SQL [,values]])`**: returns the first row of data
  - **`getCol(SQL [,column[,values]])`**: returns a single column from the data
  - **`getAssoc()`**: returns an associative array of the entire result set then frees the result set.
  - **`getAll(SQL [,values[,fetchmode]])`**: returns an array of all the rows

Example - Shortcuts

```
$when = $conn->getOne(
                "SELECT avg(pub_year) FROM books");
if (DB::isError($when)) {
  die($when->getMessage());
}
echo "The average book in the library was
                                published in $when";
```

```
list($title, $author) = $db->getRow(
    "SELECT books.title,authors.name
     FROM books, authors
     WHERE books.pub_year=1950
          AND books.authorid=authors.authorid");
echo "($title, written by $author)";
```


3.5. Metadata

- Using **`getListOf(something)`** to get information on available databases, users, views, and functions
  - something can be "**`databases`**", "**`users`**", "**`views`**", "**`functions`**".
  - E.g. $data = $conn ->getListOf("*databases*");
    - list of available databases

3.6. Transactions

- Using **`$conn->autoCommit(false)`** to set autocommit
  - Autocommit default is true
- Using **`$conn->commit()`** to commit the current transaction.
- Using **`$conn->rollback()`** to rollback the current transaction.

Example - Transactions

```
$conn->autoCommit(false);
$conn->query('CREATE TABLE blah (a integer)');
$conn->query('CREATE TABLE blue (b integer)');
$conn->commit();
$conn->query('INSERT INTO blah (a) VALUES (11)');
$conn->query('INSERT INTO blah (a) VALUES (12)');
$res = $db->query('SELECT b FROM blue');
if (DB::isError($res)) {
     echo $res->getMessage()."\n";
}
while ($res->fetchInto($row, DB_FETCHMODE_ORDERED)) {
     if ($row[0] == 12) {
          $conn->rollback();
     }
}
$res->free()
$conn->query('DROP TABLE blah');
$conn->query('DROP TABLE blue');
$conn->commit();
```


46

Question?

47