Database-enabled web technology
Summary

Instructor: Çağrı Çöltekin

c.coltekin@rug.nl

Information science/Informatiekunde

Fall 2011/12

Previous weeks

W1: Quick introductions to PHP & git.

W2: An overview of DB design and SQL.

W3: Some background on server-side programming, HTTP. Interacting with users in PHP: HTML forms, and cookies.

W4: DB Programming: stored procedures, programming with Pear DB, transactions, triggers...

W5: Session management, and a bit of security.

W6: Security...

Stored Procedures

- Stored procedures are database-side programs that are stored and run in a DBMS.
- Stored procedures add procedural-language support in relational (SQL) databases.
- Stored procedures are database objects, they are created and dropped the same way as the other database objects.
- Stored procedures run with the credentials of the user who creates them. As a result, one can run stored procedures without having access to any of the underlying tables.
- Stored procedures may reduce the network I/O, and may run faster in certain systems/cases.
- There is a relatively recent standard. However, the stored procedure language differ widely among different DBMSes.

SP in MySQL an example

```
1  drop procedure if exists confirm_order;
2  delimiter $$
3  create procedure confirm_order(in cust_id int, out nitems int)
4  begin
5      declare isbn_tmp varchar(13) default null;
6      declare customer, quantity int;
7      declare more_rows bool default true;
8      declare cur cursor for
9          select cID, ISBN, qty from basket where cID = cust_id;
10     declare continue handler for not found set more_rows = false;
11     set nitems = 0;
12     open cur;
13     fetch cur into customer, isbn_tmp, quantity;
14     while more_rows do
15         set nitems = nitems + quantity;
16         insert into orders (cID, ISBN, qty, order_date, status)
17             values (customer, isbn_tmp, quantity, now(), 'N');
18         fetch cur into customer, isbn_tmp, quantity;
19     end while;
20 end $$
21 delimiter ;
```

```
call confirm_order(10, @nbooks);
select @nbooks;
```

PHP Pear DB library

- Pear DB library provides a unified way of connecting to multiple DBMS systems from PHP.
- In comparison to other methods of database access, e.g., PHP `mysql_` functions, Pear DB provides a more portable approach.
- Independent of the DBMS or library in use, you should always validate the user input.
- Pear DB provides three functions: `escapeSimple()`, `escapeSmart()` and `quoteIdentifier()` to sanitize the input before using in an SQL statement.
- Pear DB also provides a `prepare()`/`execute()` interface (as well as the `query()`).

Pear DB: a first example

```
1  <?php
2      require_once('DB.php');
3      require_once('db-config.php');
4      $conn = DB::connect("mysql://$user:$pass@$host/$db");
5
6      $res = $conn->query('select * from book');
7
8      echo "<table border=\"1\">";
9      echo "<tr><th>ISBN</th><th>title</th></tr>";
10     while ($row = $res->fetchRow(DB_FETCHMODE_ASSOC)) {
11         echo "<tr><td>${row['ISBN']}</td>";
12         echo "<td>${row['title']}</td></tr>";
13     }
14     echo "</table>";
15     $conn->disconnect();
16 ?>
```

DB Transactions

- The (SQL) statements in a transaction is treated as atomic: either all or none of them are run.
- The (SQL) statements in a transaction is treated as isolated: DBMS isolates statements in a transaction from possible effects of other tasks running in parallel.

```
$db->autoCommit(false);
$db->query(...);
...
if (some condition) {
 $db->rollback()
} else {
 $db->commit();
}
```

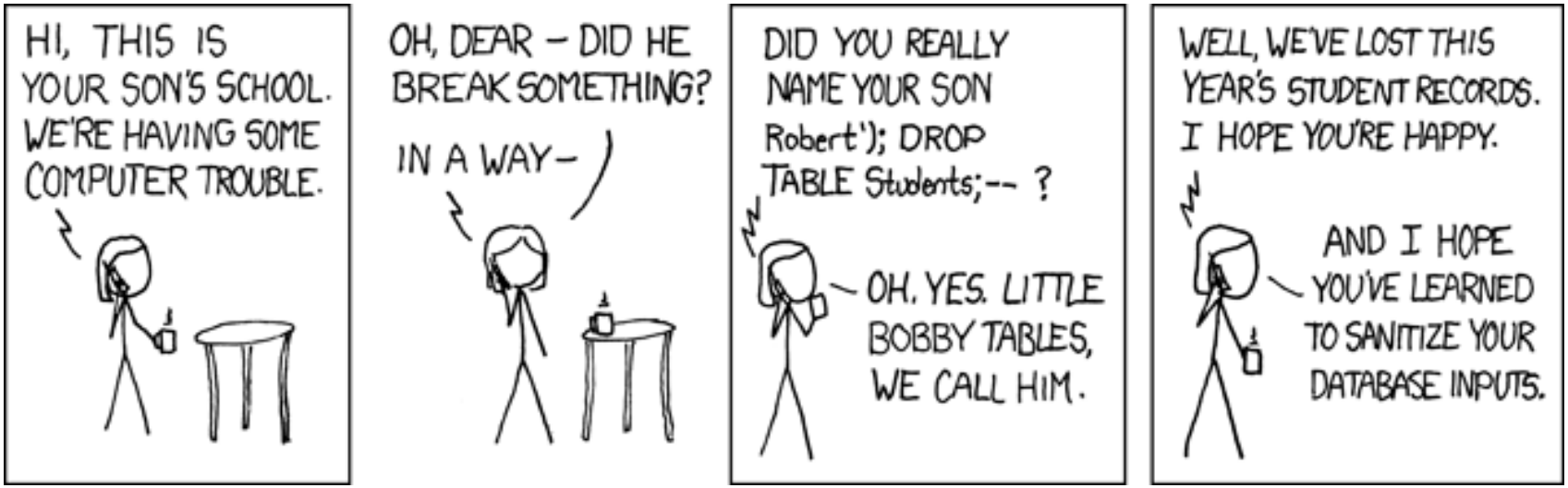
Session management: a summary

- Unlike a conventional application, a web-based application needs
  - a way to manage a user session for ensuring each execution of the process/script is originating from the same source,
  - a way to keep state during the life time of the application.
- A session consist of two components:
  - A session ID passed back-and-forth between the client an d the server.
  - A server-side storage for session data.

PHP sessions: an example

```
 1 <?php session_start(); ?>
 2 <html> <body>
 3 <?php
 4     if (!isset($_SESSION['page_seq'])) {
 5         $_SESSION['page_seq'] = 0;
 6     } else {
 7         $_SESSION['page_seq'] += 1;
 8     }
 9     echo "You are on page ${_SESSION['page_seq']}.";
10 ?>
11
12 </body></html>
```

Web, Databases & Security



http://xkcd.com/327/

OWASP 2010 top 10 web security risks

1. Injection
2. Cross-site scripting (XSS)
3. Broken authentication and session management
4. Insecure direct object references
5. Cross site request forgery (CSRF)
6. Security misconfiguration
7. Insecure cryptographic storage
8. Failure to restrict URL access
9. Insufficient transport layer protection
10. Unvalidated redirects and forwards

Sessions and Security

Badly implemented session management systems may allow unauthorized access to data/application. Typically,

- An easy to guess session ID may be found by brute-force trial & error.
- An attacker may obtain the session ID by sniffing the network traffic.
- An attacker may steal the session ID/key physically.
- An attacker may trick someone to use a URL (e.g., sent via email), causing a particular session ID to be used (session fixation).

Injection attacks: they are real

US man 'stole 130m card numbers'

US prosecutors have charged a man with stealing data relating to 130 million credit and debit cards.

Officials say it is the biggest case of identity theft in American history.

The card details were allegedly stolen from three firms, including 7-Eleven

They say Albert Gonzalez, 28, and two un-named Russian co-conspirators hacked into the payment systems of retailers, including the 7-Eleven chain.

Prosecutors say they aimed to sell the data on. If convicted, Mr Gonzalez faces up to 20 years in jail for wire fraud and five years for conspiracy.

He would also have to pay a fine of $250,000 (£150,000) for each of the two charges.

'Standard' attack

Mr Gonzalez used a technique known as an "SQL injection attack" to access the databases and steal information, the US Department of Justice (DoJ) said.

The method is believed to involve

SQL INJECTION ATTACK

- This is a fairly common way that fraudsters try to gain access to consumers' card details.
- They scour the internet for weaknesses in companies' programming which allows them to

`http://news.bbc.co.uk/2/hi/americas/8206305.stm` (2009-09-18)

Injection attacks: they are real

US man 'stole 130m card numbers'

US prosecutors have charged a man with stealing data relating to 130 million credit and debit cards.

Officials say it is the biggest case of identity theft in American history.

They say Albert Gonzalez, 28, and two un-named Russian co-conspirators hacked into the payment systems of retailers, including the 7-Eleven chain.

The card details were allegedly stolen from three firms, including 7-Eleven

Prosecutors say they aimed to sell the data on. If convicted, Mr Gonzalez faces up to 20 years in jail for wire fraud and five years for conspiracy.

He would also have to pay a fine of $250,000 (£150,000) for each of the two charges.

'Standard' attack

Mr Gonzalez used a technique known as an "SQL injection attack" to access the databases and steal information, the US Department of Justice (DoJ) said.

The method is believed to involve

SQL INJECTION ATTACK

- This is a fairly common way that fraudsters try to gain access to consumers' card details.
- They scour the internet for weaknesses in companies' programming which allows them to

`http://news.bbc.co.uk/2/hi/americas/8206305.stm` (2009-09-18)

- (SQL) injection attacks are prevalent, even in cases where people take security seriously.
- A simple mistake in the code can make large investments to computer security useless.
- Consequences of the vulnerability may differ.
- It is easy to prevent: never trust user input.

Cross-site scripting (XSS)

XSS attacks come in many shapes and sizes, but in it is essence: attacker tricks user/browser to run a script while viewing another site.
A typical case:

1. Attacker plants the malicious script (e.g., using SQL injection) to a legitimate web site.
2. Victim visits the web-site, running the script in the context of the web site.
3. Script sends valuable (e.g., session credentials) to the attacker.

Solution to fix most security problems

Sanitize your input (and output too).

Solution to fix most security problems

Sanitize your input (and output too).

If you are using PHP:

SQL Use, for example, `DB::escapeSimple()`, or `prepare()`/`execute()` instead of `query()`.

shell Use `escapeshellarg()` or `escapeshellcmd()`.

HTML Use `htmlspecialchars()` while writing HTML code literally on a web page (particularly against XSS).

others If you are implementing your own escape/validation routine, use white listing: explicitly say what you accept, instead of what you do not.

A few guidelines (before we start)

- Always check user input before using (e.g., in an SQL query).
- Do not store and transfer sensitive information unencrypted.
- Do not store or transfer sensitive information if you can avoid it.
- Sanitize your output (e.g., properly escape special characters if you are outputting HTML).
- Try to implement multiple levels/layers of security.

Today...

- An overall summary.
- A few notes on passwords.

How (not) to store and use passwords

- Do not store passwords in clear.

How (not) to store and use passwords

- Do not store passwords in clear.
- Always transfer passwords (and other sensitive information) via an encrypted connection.

How (not) to store and use passwords

- Do not store passwords in clear.
- Always transfer passwords (and other sensitive information) via an encrypted connection.
- Storing hashes (e.g., MD5, SHA-256, ...), of passwords does the same job (most of the time).

How (not) to store and use passwords

- Do not store passwords in clear.
- Always transfer passwords (and other sensitive information) via an encrypted connection.
- Storing hashes (e.g., MD5, SHA-256, ...), of passwords does the same job (most of the time).
- Use multiple hashing, and salts.

How (not) to store and use passwords

- Do not store passwords in clear.
- Always transfer passwords (and other sensitive information) via an encrypted connection.
- Storing hashes (e.g., MD5, SHA-256, ...), of passwords does the same job (most of the time).
- Use multiple hashing, and salts.
- If you think you have to store passwords, think again.

How (not) to store and use passwords

- Do not store passwords in clear.
- Always transfer passwords (and other sensitive information) via an encrypted connection.
- Storing hashes (e.g., MD5, SHA-256, ...), of passwords does the same job (most of the time).
- Use multiple hashing, and salts.
- If you think you have to store passwords, think again.
- If you really have to store passwords, code them, e.g., using base 64, while storing. (This is only a protection against unintentional viewing.)

Hash functions

A (cryptographic) hash function maps an arbitrary length data to a fixed-length bit string.

- A hash function must be deterministic: given the same data it has to return the same hash value.
- It is difficult (computationally infeasible) to generate the data given the hash value.
- Multiple data streams may have the same hash function, but a good algorithm reduces the likelihood of collisions.

Using hash functions in PHP

The function `hash()` provides a uniform interface for many hash algorithms.

```
1 $pwdhash = hash('sha256', $_REQUEST{'password'});
2 $qres = db->query("select from user "
3           . "where username = '"
4           .  db->escapeSimple($_REQUEST['user']) . "'"
5           . "and password = '" . $pwdhash . "'");
6 if ($qres->numRows() == 1 ) {
7     // login ok
8 ...
```

`hash_algos()` return available hash algorithms.

Note that you still need to make sure that the password is not sent over network unencrypted.

Passwords can be 'cracked'

- If someone obtains the hash values, they cannot calculate the passwords.
- But, they can test it against a large number of strings (e.g., from a dictionary).
- This attack becomes more effective, if the attacker pre-computes the hash values for these strings.

Salting and multiple hashing

Against password cracking:

- A strategy to make this difficult is called salting:
  You pick a random string, 'the salt', and combine it with the password before hashing:
  ```
  $phash = $salt . hash($algo, $pwd . $salt);$
  ```
  The attacker has to pre-compute and store hashes for all possible salts.
- Another method is multiple hashing:
  ```
  hash($algo, hash($algo, $str))
  ```
  This makes the computation slower. It's OK for checking once in a while, but it's a burden if you try to compute millions of them.

Passwords can be 'guessed'

- An attacker may try user names and passwords on the login page of your application.
- Generally, the attacker will first guess the valid user names.
- Nest, the attacker may try a dictionary attack for the passwords.

Common precautions:

- The system should not respond differently to valid and unknown users.
- To many successive login attempts should be prevented.
  - disable the account after some number of unsuccessful attempts,
  - slow down login response (exponentially) for each unsuccessful attempt.

Project evaluation

- Good database design
- Secure and efficient web programming
- You will need to submit two reports:
  - Initial design report (3 to 8 pages), which should include,
    - a summary of the project requirements
    - your Initial database design (e.g., E-R design, DB schema)
    - type(s) of users, typical queries expected, security (authentication, authorization) requirements.
  - You will get feedback on your design based on this report.
  - Final report (5 to 10 pages), which can include the some of the above, but also,
    - a brief 'user guide'
    - final state of the project and possible future work
  - Clarity of the reports will contribute to your final score.
  - All project documentation has to be in English, but you will not loose points for language mistakes.

About the exam

- Some SQL/DBMS questions.
- Programming related (PHP).
- Web programming (forms, cookies, ...)
- Security.
- Short-answer and multiple-choice questions to be easy on your memory.
- You should not be worried if you are working on your projects.

XSS example: blog display—full source

```
1  <?php
2      session_start();
3      require_once('DB.php');
4      require_once('blog-db-conf.php');
5      $db = DB::connect("$dbspec");
6
7      if (PEAR::isError($db)) {
8          echo $db->getMessage();
9      }
10
11     $res = $db->query('select * from posts;');
12     while ($row = $res->fetchRow(DB_FETCHMODE_ASSOC)) {
13         echo "<p>${row['text']}";
14     }
15 ?>
```

XSS example: blog post—full source

```
 1 <?php
 2     session_start();
 3     require_once('DB.php');
 4 if (isset($_REQUEST['submit'])){
 5     require_once('blog-db-conf.php');
 6     $db = DB::connect("$dbspec");
 7
 8     $q = $db->prepare("insert into posts values(0,?);");
 9     $text = $_REQUEST['post'];
10     $res = $db->execute($q, $text);
11 }
12 ?>
13
14 <form  method="post"
15        action="<?php echo "${_SERVER['PHP_SELF']}";?>">
16 Post: <input type="text" name="post"/><br>
17           <input type="submit" name="submit" value="submit">
18 </form>
```