WHERE ARE WE NOW?

W34: Introduction, DBMSs and Relational Databases
W35: Developing Database Systems
W36: SQL –Part I
W37: SQL –Part II and Relational Algebra
W38: Data Modelling
W39: Data Modelling
W40: Database Design
W41: Normalisation and Stored Procedures
W42: XML and Web Technology
W43: Processing XML Data
W44: XML Validation
W45: Beyond relational databases and XML
W46: File Organisations and Indexes
W47: Database Security and Administration
W48: Transaction Processing and Wrap-up

INFO

- New resource link added on Fronter:
  - *Webucator [XML Schema Tutorial: XML Schema Keys]*
- Labs/teaching assistants this week and next:
  - *Are you interested in exam question assistance?*
  - *Or should we just cancel the remaining lab sessions?*
- Next week's program

GOAL

- Today:
  - *PHP/PDO revisit*
  - *Security in web database application*
  - *Database privileges*
- Wednesday:
  - *Database administration*
  - *Database backup and recovery*
  - *Concurrency and database transactions*

PHP

- Short program-test-program-test... loops
- PHP vs HTML
- Using standard libraries rather than developing your own:
  - *String handling*
  - *SAX/DOM/XPATH/XSLT*
  - *Style guides:*
    - » http://pear.php.net/manual/en/standards.php

DATABASE PROGRAMMING IN PHP (1)

- Pseudocode for accessing database data:

```
BEGIN
  Connect to database
  Create prepared statement
  Bind parameters
  Execute query
  FOR EACH row
    Retrieve values
    Process row
  END FOR
END
```

DATABASE PROGRAMMING IN PHP (2)

- Use PDO – not mysqli:
  - *Implemented by several DB vendors*
  - *Supports named parameters*
- Use prepared statements
- PDOStatement::fetch()/::fetchAll():
  - *Represents a database cursor*
  - *Returns (by default) columns as a numeric array and as an associative array*

CONTENT

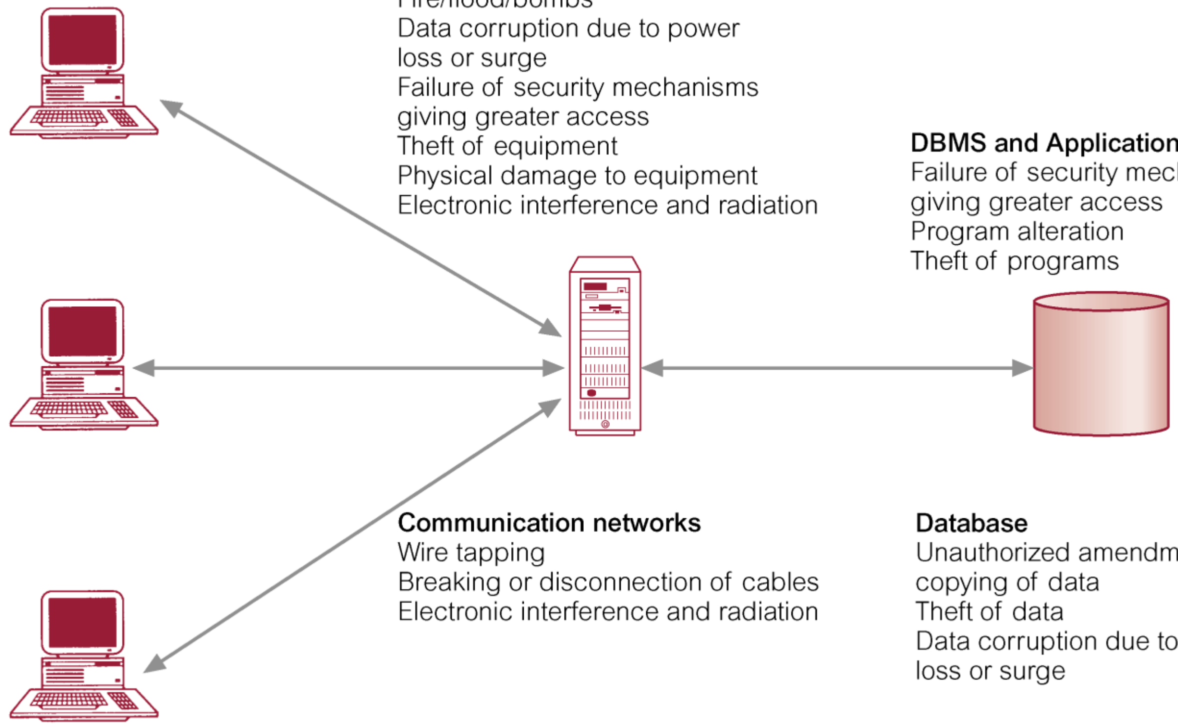
- Database security:
  - *Threats and countermeasures*
- Security in database web applications:
  - *Robust programming*
  - *Secure HTTP*
  - *Passwords*
  - *SQL injection*
- Privileges and roles
- Security in MySQL

SITUATIONS THAT MAY CAUSE A THREAT TO THE DB

- Theft and fraud
- Loss of confidentiality (secrecy)
- Loss of privacy
- Loss of integrity
- Loss of availability

Hardware
Fire/flood/bombs
Data corruption due to power loss or surge
Failure of security mechanisms giving greater access
Theft of equipment
Physical damage to equipment
Electronic interference and radiation

DBMS and Application Software
Failure of security mechanism giving greater access
Program alteration
Theft of programs



Communication networks
Wire tapping
Breaking or disconnection of cables
Electronic interference and radiation

Database
Unauthorized amendment or copying of data
Theft of data
Data corruption due to power loss or surge

Users
Using another person's means of access
Viewing and disclosing unauthorized data
Inadequate Staff training
Illegal entry by hacker
Blackmail
Introduction of viruses

Programers/Operators
Creating trapdoors
Program alteration (such as creating software that is insecure)
Inadequate staff training
Inadequate security policies and procedures
Staff shortages or strikes

Data/Database Administrator
Inadequate security policies and procedures

COMPUTER-BASED COUNTER-MEASURES

- Authentication and authorisation
- Access control
- Views
- Backup and recovery
- Integrity
- Encryption
- RAID technology

CONTENT

- Database security:
  - *Threats and countermeasures*
- Security in database web applications:
  - *Robust programming*
  - *Secure HTTP*
  - *Passwords*
  - *SQL injection*
- Privileges and roles
- Security in MySQL

ROBUST PROGRAMMING (1)

- Test values before using them:

```
if ($doc->getElementsByTagName('title')->size > 0) {
  echo $doc->getElementsByTagName('title')->item(0);
}
```

- Prefer foreach:

```
foreach ($doc->getElementsByTagName('title') as $el) {
  // Process element
}
```

- Rather than:

```
for($i = 0;
    $i < $doc->getElementsByTagName('title')->size;
    $i++ ) {
 $el = $doc->getElementsByTagName('title');
  // Process element
}
```

ROBUST PROGRAMMING (2)

- Never trust what comes from the client:
  - *Users may – deliberately or by mistake – pass invalid data*
  - *You never know whether field verification code has actually been run within the browser*
  - *Use APIs (such as the DOM API) rather than direct data manipulation:*
    - » But make sure to keep up to date on the quality of the API code
  - *Choose the safer alternative:*
    - » E.g. prepared statements rather than direct SQL statements
- Never disclose programming details to the end-user:
  - *Replace API error messages with application specific ones*

WHAT MAY HAPPEN IF WE TRUST USER DATA

- Assume that the end-user creates a blog entry by filling in a form (cf Oblig 1):
  - *What if the user enters this as the blog text:*

    ```
    <script>document.body.style.visibility='hidden'
    </script>
    ```
  - *Allowing this kind of input will allow the hackers easy ways to stop a service – possibly without the service provider noticing*

SECURE HTTP

- SSL (Secure Socket Layer) can be utilised to encrypt communication between client and server:
  - *URL:*
    - » `HTTPS://...`
  - *PHP does not deal directly with SSL, but can handle HTTPS requests differently from plain requests:*

```
if ($_SERVER['HTTPS'] !== 'on') {
  die("Must be a secure connection.");
}
```

ENCRYPTION AND HASHING

- Encryption – decryption:
  - *Text can be stored in encrypted form and decrypted after being retrieved*
  - *PHP mcrypt functions*
- Hashing – a one-way encryption:
  - *Impossible to decode*
  - *Useful for checking the correctness of a given value:*
    - » Authentic document
    - » Correct password
    - » …
  - *MD5, SHA1, …*
  - *But a rainbow table may contain hashes for millions of strings*

DATABASE PASSWORDS

- The PHP source files may be comprimised:
  - *Automatic backup files, e.g.,* `.php~`
  - *File download enabled, e.g.,* `'download.php?index.php'`
- Store database password in configuration files is preferable

USER-PROVIDED PASSWORDS

- Never store plain text
- Use a strong and slow hashing function
- Add a "salt" to the user-provided password:

```
$pwd = crypt($uPwd, '$2a$07$');
```

- Use an external authentication service?
  - *Such as LDAP, Active Directory*

SQL INJECTION

- How can it happen?
- Examples of what may happen
- How to avoid

SQL INJECTION - HOW CAN IT HAPPEN?

- Exploiting SQL meta-characters:
  - *' – end of string*
  - *\ – escape of special character*
  - *; – statement delimiter*
  - *-- – start of comment*
- Client input passed directly to the database
- SQL query structure

SQL INJECTION EXAMPLE (1)

- Assume a web page listing CDs in a music shop:
  - *URL:*
    ```
    cdListing.php?genre=Pop
    ```
  - *Passed to the select statement:*
    ```
    SELECT title, artist FROM CD
      WHERE genre='$genre'
    ```
- Assume that the user passes:
  ```
  Pop' union select name, password from user; --
  ```
  - *What happens now?*
    ```
    SELECT title, artist FROM CD
      WHERE genre='Pop' union select name, password
                   from user; -- '
    ```

SQL INJECTION EXAMPLE (2)

- Assume that the user passes:

  `Pop'; drop table user; --`

  - *What happens now?*

```
SELECT title, artist FROM CD
  WHERE genre='Pop'; drop table user; -- '
```

SQL INJECTION EXAMPLE (3)

- Assume that the user passes:

```
Pop'; update user set password='new' where
name='runehj'; --
```

- *What happens now?*

```
SELECT title, artist FROM CD
  WHERE genre='Pop'; update user
               set password='new'
               where name='runehj'; -- '
```

SQL INJECTION EXAMPLE (4)

- Assume that user name and password is used for logon:
  - *URL:*

    ```
    login.php?name=runehj&pwd=aPwd
    ```

  - *Passed to the select statement:*

    ```
    SELECT * FROM user
      WHERE name='$name' AND password='$pwd'
    ```

- Assume that the user passes:

  ```
  runehj' or '1'='1
  ```

  - *What happens now?*

    ```
    SELECT * FROM user
      WHERE name='runehj' or '1'='1'
      AND password='$pwd'
    ```

SQL INJECTION - HOW TO AVOID

- Escape/reject any character that is not to be expected:
  - *Digits only in numbers*
  - *Valid email characters only in characters*
  - *Valid name characters only in names*
  - …
- Use prepared statements
- Avoid providing the hacker any help and hints

CONTENT

- Database security:
  - *Threats and countermeasures*
- Security in database web applications:
  - *Robust programming*
  - *Secure HTTP*
  - *Passwords*
  - *SQL injection*
- Privileges and roles
- Security in MySQL

DATABASE PRIVILEGES (1)

- A privilege allows a user to manage:
  - *Operations on the database:*
    - » `CREATE USER`
    - » `SHUTDOWN`
    - » `...`
  - *Operations on individual database objects:*
    - » `CREATE`
    - » `DROP`
    - » `ALTER`
    - » `DELETE`
    - » `INSERT`
    - » `SELECT`
    - » `UPDATE`
    - » `...`

DATABASE PRIVILEGES (2)

- Privileges:
  - *Only administrator should be granted admin privileges*
  - *The database should not run as user root*
  - *The web app database user should not be the admin user*
- Privileges can be granted to users:

  -
    ```
    GRANT SELECT, INSERT
          ON imt2571.*
          TO 'astudent'@'localhost'
    ```
- And can be revoked from users:

  -
    ```
    REVOKE INSERT
           ON imt2571.*
           FROM 'astudent'@'localhost';
    ```

DATABASE ROLES

- A role is a named group of privileges
- Individual user can be granted one or more roles:
  - *Thereby granted the roles' privileges*
- Roles allow for easier and better management of privileges
  - *Privileges should preferably be granted to roles*

SECURITY IN MySQL

- Encryption and hashing:
  - *Mcrypt functions:*
    - » `mcrypt_encrypt()`
    - » `mcyprt_decrypt()`
    - » `...`
  - *Hashing functions:*
    - » `crypt()`
    - » `md5()`
    - » `sha1()`
    - » `...`
- No support for database roles:
  - *Privileges are granted to individual users*

RESOURCES

- C&B 7.6, 20–20.2, 20.5
- PHP Manual, Database Security og Safe Password Hashing
- MySQL Manual, 6.2 The MySQL Access Privilege System
- Steve Friedl, SQL Injection Attacks by Example
- alias.io, How to store passwords safely with PHP and MySQL