M183 Applikationssicherheit Implementieren
12

By Jürg Nietlispach

Recap # 11

Authorization

- Data Access Control
  - Actors, Actions
  - MAC, DAC, RBAC, Hybrid Models, 3x3 Matrix

- Permission Models
  - Read, Write, Execute

Data Access Control – what's next?

So far: Data Access Control defines i.e. whether an actor can access certain data or not.

Now: In case an actor has access to certain data, how is the data access achived?

Data Access

Data?

- Files, Documents, Images (Binary, Blob, etc.)
- Passwords, Usernames, Permissions, Settings
- Text, JSON, XML, HTML
- Geolocations, Protocols
- …

Where is it stored & How to access it (CRUD)?

- **Databases**
    - Key-Value (MongoDB), Timestamp-Based (InfluxDB), **Relational (MySQL)**

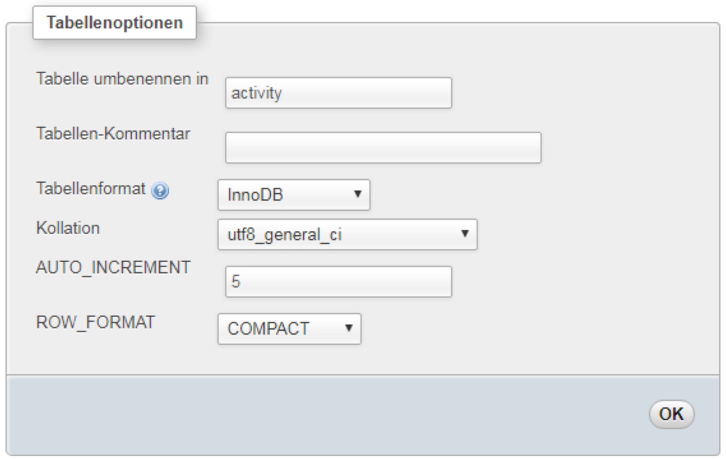
- Data-Ressources via **HTTP**
    - Images, Documents, Zip-Files etc.

- …

Data Management with (Relational) Databases

How Data is stored

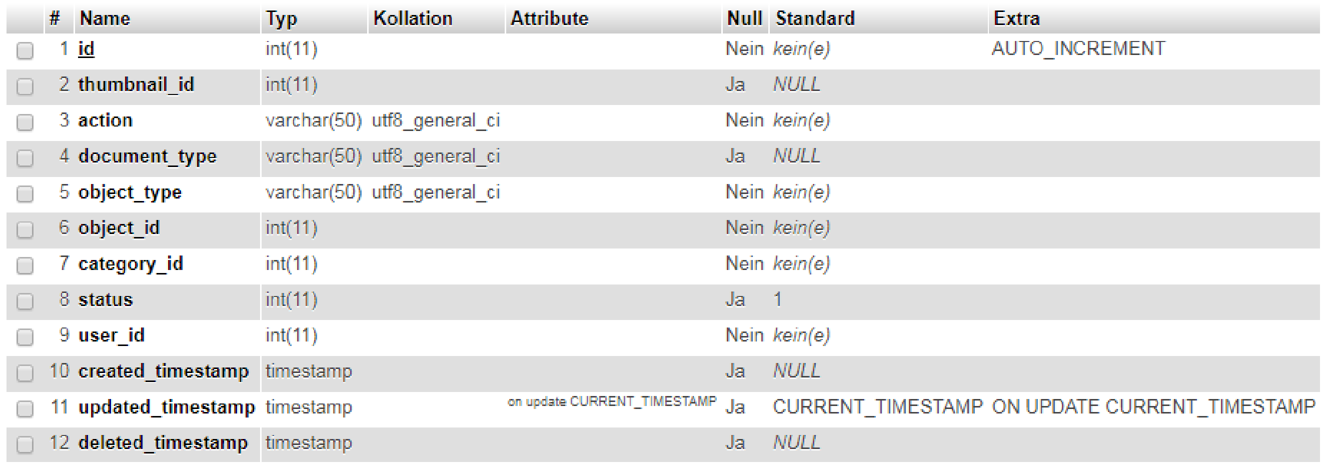
Store the Data in Table-Format where every
- table has it’s own collation and character set
- column has it’s own datatype
- row contains the specific data



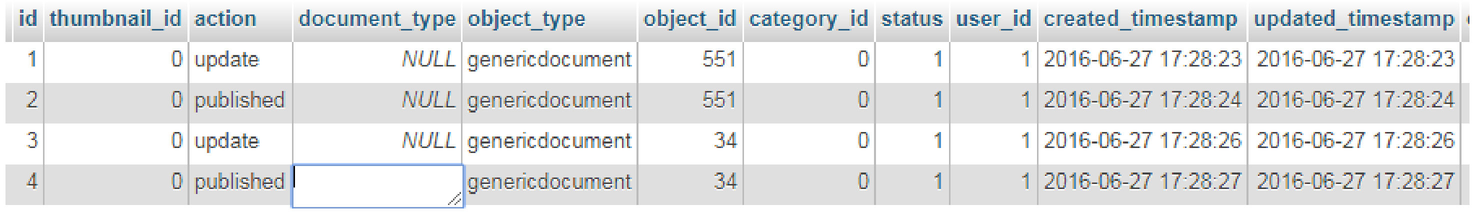
Tabellenoptionen

Tabelle umbenennen in	activity
Tabellen-Kommentar	
Tabellenformat	InnoDB
Kollation	utf8_general_ci
AUTO_INCREMENT	5
ROW_FORMAT	COMPACT

OK



#	Name	Typ	Kollation	Attribute	Null	Standard	Extra
1	id	int(11)			Nein	*kein(e)*	AUTO_INCREMENT
2	thumbnail_id	int(11)			Ja	*NULL*	
3	action	varchar(50)	utf8_general_ci		Nein	*kein(e)*	
4	document_type	varchar(50)	utf8_general_ci		Ja	*NULL*	
5	object_type	varchar(50)	utf8_general_ci		Nein	*kein(e)*	
6	object_id	int(11)			Nein	*kein(e)*	
7	category_id	int(11)			Nein	*kein(e)*	
8	status	int(11)			Ja	1	
9	user_id	int(11)			Nein	*kein(e)*	
10	created_timestamp	timestamp			Ja	*NULL*	
11	updated_timestamp	timestamp		on update CURRENT_TIMESTAMP	Ja	CURRENT_TIMESTAMP	ON UPDATE CURRENT_TIMESTAMP
12	deleted_timestamp	timestamp			Ja	*NULL*	

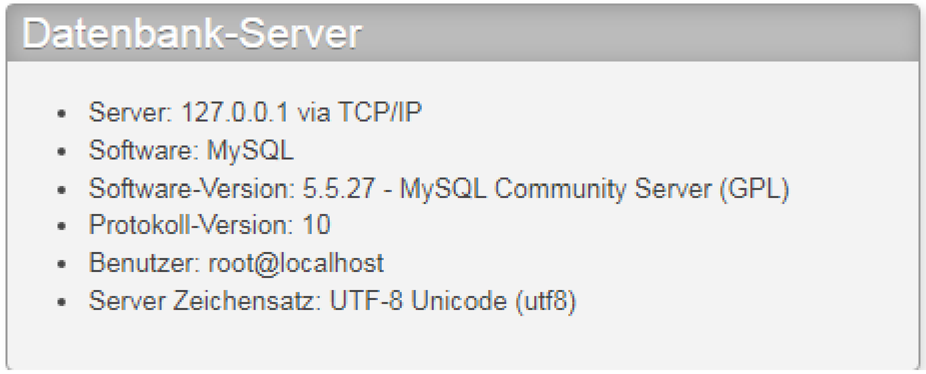


id	thumbnail_id	action	document_type	object_type	object_id	category_id	status	user_id	created_timestamp	updated_timestamp
1	0	update	*NULL*	genericdocument	551	0	1	1	2016-06-27 17:28:23	2016-06-27 17:28:23
2	0	published	*NULL*	genericdocument	551	0	1	1	2016-06-27 17:28:24	2016-06-27 17:28:24
3	0	update	*NULL*	genericdocument	34	0	1	1	2016-06-27 17:28:26	2016-06-27 17:28:26
4	0	published		genericdocument	34	0	1	1	2016-06-27 17:28:27	2016-06-27 17:28:27

Data Management with Databases - Access

Connection to Data

- Connection over TCP/IP (with or without SSL) on dedicated port (3306)
- With username & password
- Connection-String Collation



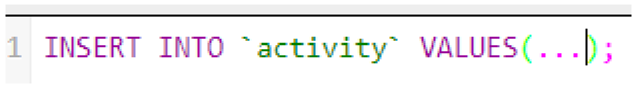
Datenbank-Server

- Server: 127.0.0.1 via TCP/IP
- Software: MySQL
- Software-Version: 5.5.27 - MySQL Community Server (GPL)
- Protokoll-Version: 10
- Benutzer: root@localhost
- Server Zeichensatz: UTF-8 Unicode (utf8)

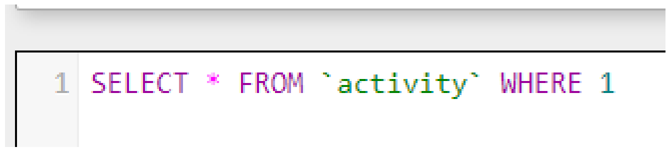
Data Management with Databases - CRUD

How Data is managed

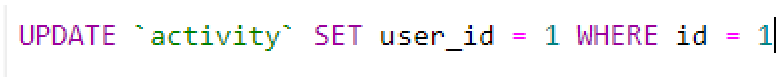
- SQL-Language (Structured Query Language) for Create, Read, Update & Delete (CRUD-Operations)



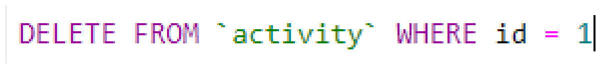
```sql
INSERT INTO `activity` VALUES(...);
```



```sql
SELECT * FROM `activity` WHERE 1
```



```sql
UPDATE `activity` SET user_id = 1 WHERE id = 1
```



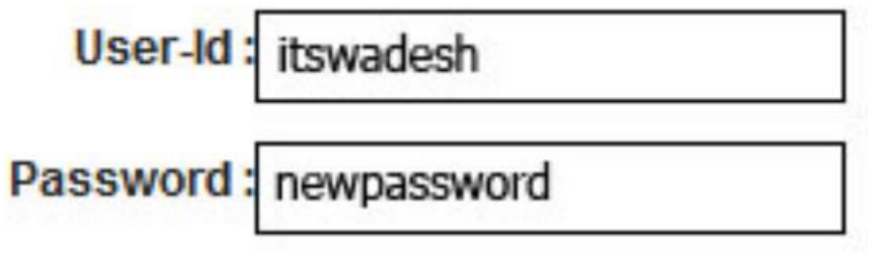
```sql
DELETE FROM `activity` WHERE id = 1
```

Database Attacks?

- **SQL-Injections**
- **Missing Input Filtering & Validation (Text-Fields)**
- **Privilege Abuse & Privilege elevation**
- **Stored Procedures**
- DoS
- …

SQL-Injections

Normal-Szenario



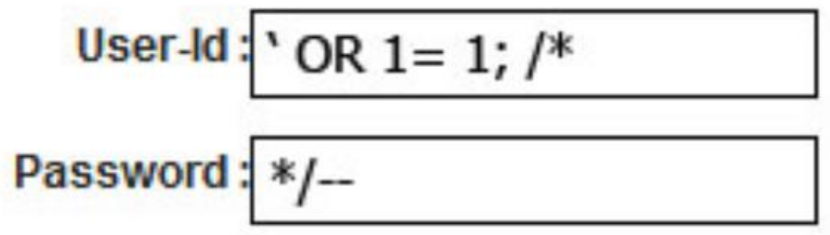
User-Id : itswadesh

Password : newpassword

```
String query = "SELECT * FROM accounts WHERE
username='" +
request.getParameter(“username") + "‘ AND
password=‘" +
request.getParameter(“password") + "’";
```

SQL-Injections 2

Attack-Szenario



User-Id : ‘ OR 1= 1; /*

Password : */--

```
String query = "SELECT * FROM accounts WHERE
username=‘’ OR 1=1 /* ‘ AND password=‘*/--’”;
```

```
String query = "SELECT * FROM accounts WHERE
username=‘’ OR 1=1”;
```

Alle Accounts (inkl. allen Feldern der accounts-Tabelle) werden retourniert!

SQL-Injection Prevention

Manual Escape (Single) Quotes:

```
String query = "SELECT * FROM accounts WHERE username='\' OR 1=1 /* ' AND
password='*/--'";
```

⇒ Results in a parse error (Caution: do not expose DB-Credentials and fail silently/securely)

Filter Parameter (from MYSQL-Keywords, HTML, JS-Tags)

SQL-Injection Prevention 2

Manual Filter Parameter (MYSQL-Keywords)

```
1. ’ OR 1=1 /* and  */-- eliminated:

String query = "SELECT * FROM accounts WHERE username=‘‘ AND password=‘’”;
```

Prepared Statements

Using an ORM:

```
$sql = 'SELECT name, colour, calories
FROM fruit  WHERE calories < :calories AND colour = :colour';

$sth = $dbh->prepare($sql);

$sth->execute(array(':calories' => 150, ':colour' => 'red'));

$red = $sth->fetchAll();
```

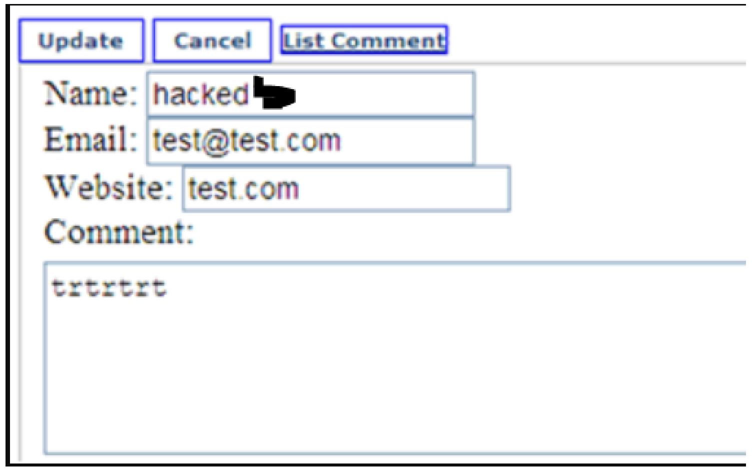
Prepared Statements 2

Using an ORM:

```
$sth = $dbh->prepare('SELECT name, colour, calories FROM fruit
WHERE calories < ? AND colour = ?');

$sth->execute(array(150, 'red')); // calories -> 150, colour -> ‘red’

$red = $sth->fetchAll();
```

Input Filtering

Normal-Szenario



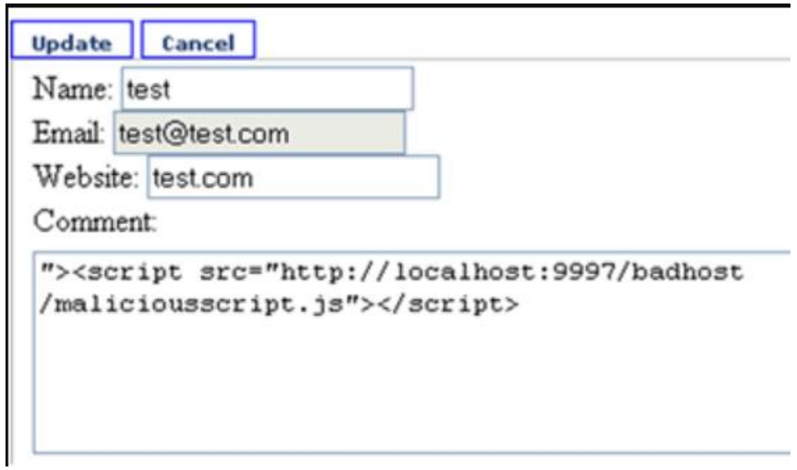
Name: hacked
Email: test@test.com
Website: test.com
Comment:
trtrtrt

```
String query = “INSERT INTO comments SET(name,
email, website, comment) VALUES (‘”+
request.getParameter(“name") +”’,’ ”+
request.getParameter(“email") +”’,’ ”+
request.getParameter(“website") +”’,’ ”+
request.getParameter(“comment") +”’);”
```

```
String query = “INSERT INTO comments SET(name,
email, website, comment) VALUES (‘Hans Muster’,
’hans@muster.ch’, ’https://muster.ch’, ’Sehr
schöne Webseite’);”
```

Input Filtering 2

Attack-Szenario



Name: test
Email: test@test.com
Website: test.com
Comment:

```
"><script src="http://localhost:9997/badhost
/maliciousscript.js"></script>
```

```
String query = “INSERT INTO comments SET(name,
email, website, comment) VALUES (‘”+
request.getParameter(“name") +”’,’ ”+
request.getParameter(“email") +”’,’ ”+
request.getParameter(“website") +”’,’ ”+
request.getParameter(“comment") +”’);”
```

```
String query = “INSERT INTO comments SET(name,
email, website, comment) VALUES (‘…’, ’…’, ’…’,
’<script> XSS – Script </script>’);”
```

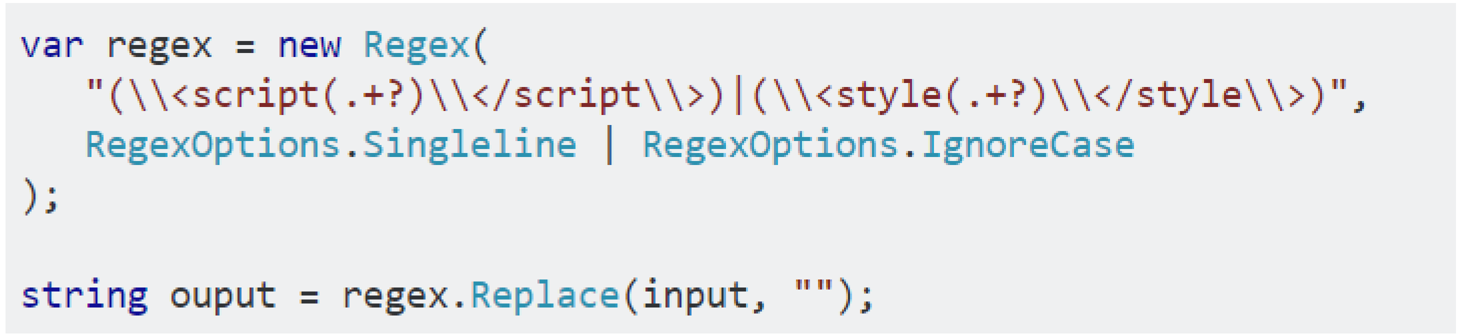
How to Filter the Input 1

Framework Functionality (like strip_tags(), usage of HtmlDocument etc.)

```
HtmlDocument doc = new HtmlDocument();
doc.LoadHtml(htmlInput);

var nodes = doc.DocumentNode.SelectNodes("//script|//style");

foreach (var node in nodes)
    node.ParentNode.RemoveChild(node);

string htmlOutput = doc.DocumentNode.OuterHtml;
```

How to Filter the Input 2

(Custom) Regular Expressions

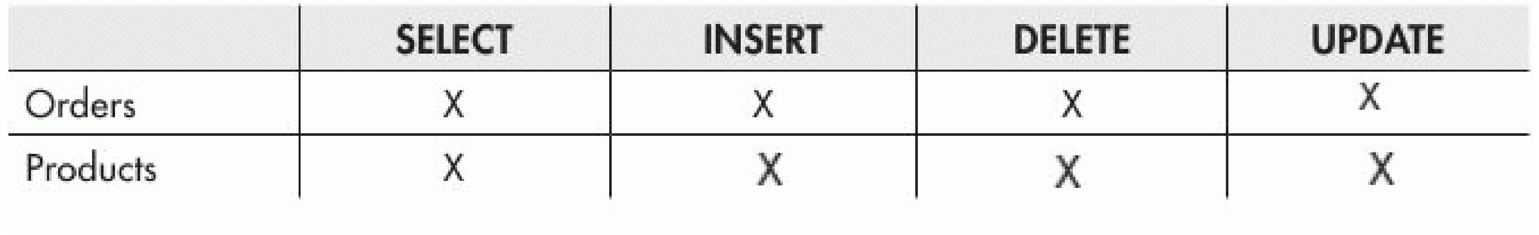


```
var regex = new Regex(
   "(\\<script(.+?)\\</script\\>)|(\\<style(.+?)\\</style\\>)",
   RegexOptions.Singleline | RegexOptions.IgnoreCase
);

string ouput = regex.Replace(input, "");
```

Database Permissions

«Normal» Case
Database Accessed through one single user for all users of a web application

Example
User «John Doe», etc. accesses the database with the «databaseadmin» username which has full privilege



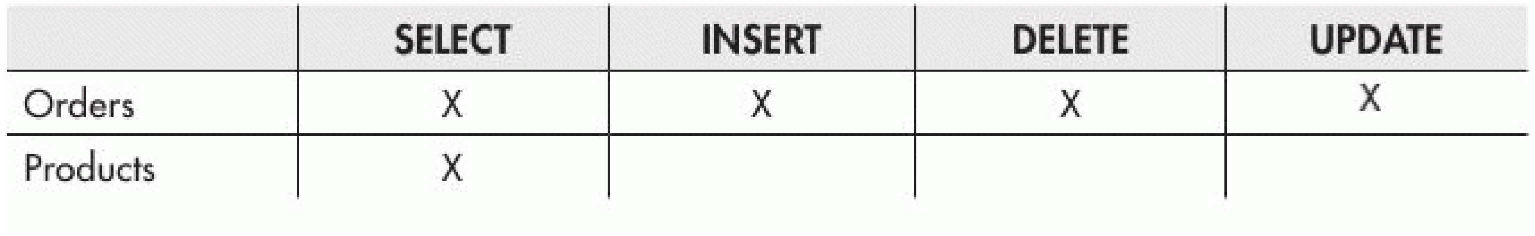
	SELECT	INSERT	DELETE	UPDATE
Orders	X	X	X	X
Products	X	X	X	X

Database Permissions 2

«Improved» Case
Database Accessed through **many user(-roles)** for **certain** users of a web application

Example
User «John Doe», etc. accesses the database with the «web_app_user» username which has only full privilege on the orders table using:

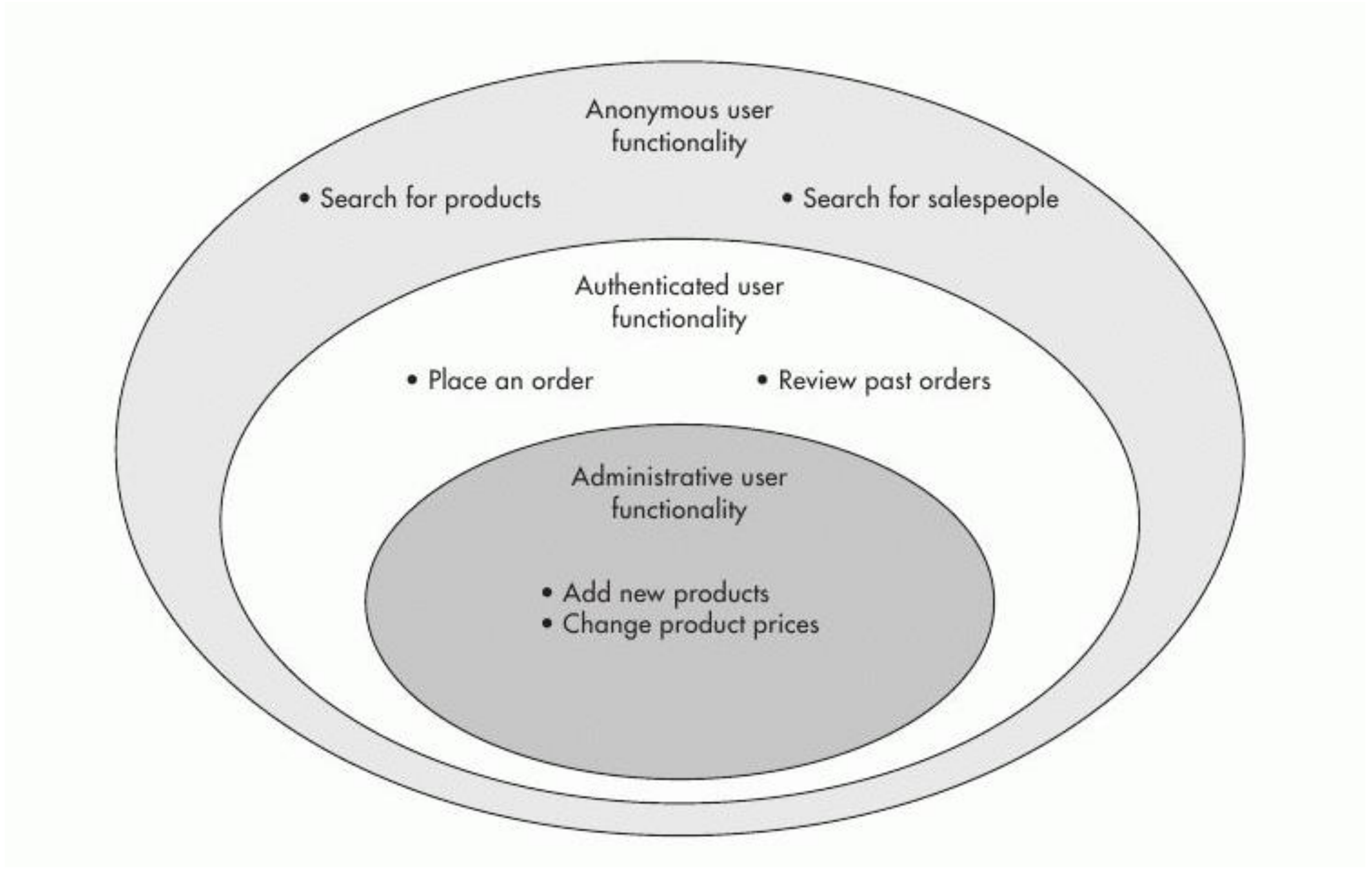
```
REVOKE UPDATE ON Products FROM web_app_user
REVOKE DELETE ON Products FROM web_app_user
REVOKE INSERT ON Products FROM web_app_user
```



	SELECT	INSERT	DELETE	UPDATE
Orders	X	X	X	X
Products	X			

Database Permissions 3

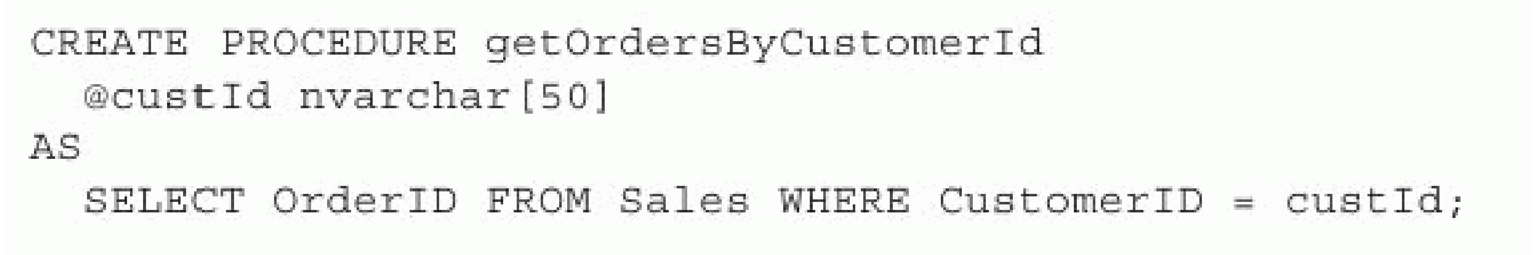
Example Roles & Permissions



Stored Procedures

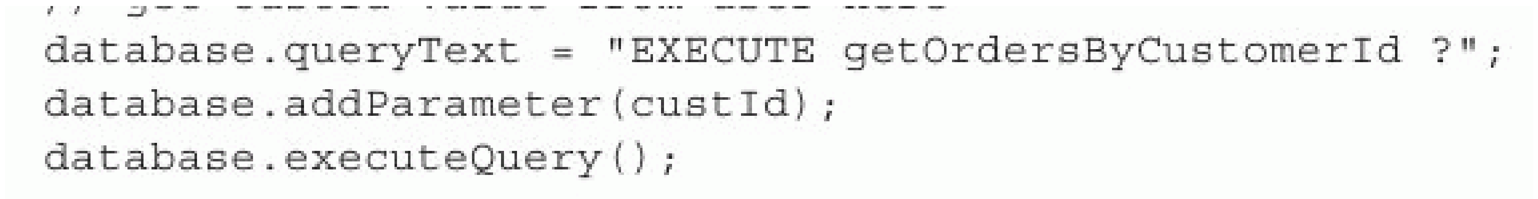
Idea: to grant a user only the permission to execute stored procedures (added only by a db-admin for instance) only! **Thus:** No Insert, Update, Delete, Select operation possible for the user!

Example
On the Database directly, create a stored query:



```
CREATE PROCEDURE getOrdersByCustomerId
  @custId nvarchar[50]
AS
  SELECT OrderID FROM Sales WHERE CustomerID = custId;
```

Execute it like so



```
[illegible]
database.queryText = "EXECUTE getOrdersByCustomerId ?";
database.addParameter(custId);
database.executeQuery();
```

Caution: Injections!

Data Management with Data Ressources

Static Content vs Dynamic Data

Static – Delivered «as is»
- JPEG, BMP, PNG
- HTML
- .mp3, .mp4

Index.html:

```
<html>
  <body>
    <h1>Welcome to Dave's photo gallery</h1>
    <img src="images/whistler_vacation.jpg" />
    <img src="images/eagle_on_bike_trail.jpg" />
    <img src="images/kitty_napping.jpg" />
    ...
  </body>
</html>
```

Dynamic – Delivered after Processing
- .aspx
- .php
- .jsp

Index.php:

```
<html>
  <body>
    <h1>Random photo from Dave's photo gallery</h1>
    <?php
      $allImages = glob("images/*.jpg");
      $randomImage = $allImages[array_rand($allImages, 1)];
      echo "<img src=\"" . $randomImage . "\" />";
    ?>
  </body>
</html>
```

Data Ressources – Upload

HTML Form using enctype

```
<form enctype="multipart/form-data" action="http://localhost:3000/upload?upload_progress_id
<input type="hidden" name="MAX_FILE_SIZE" value="100000" />
Choose a file to upload: <input name="uploadedfile" type="file" /><br />
<input type="submit" value="Upload File" />
</form>
```

Is translated to the following POST - Request

```
POST /upload?upload_progress_id=12344 HTTP/1.1
Host: localhost:3000
Content-Length: 1325
Origin: http://localhost:3000
... other headers ...
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryePkpFF7tjBAqx29L

------WebKitFormBoundaryePkpFF7tjBAqx29L
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
------WebKitFormBoundaryePkpFF7tjBAqx29L
Content-Disposition: form-data; name="uploadedfile"; filename="hello.o"
Content-Type: application/x-object
```

Data Ressources – Download

HTTP – Request to a Ressource

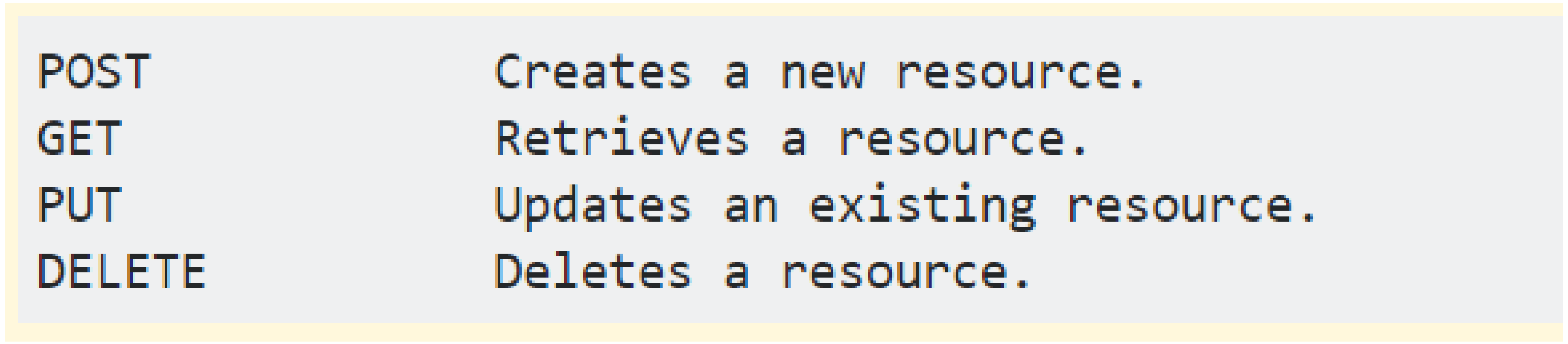
```
<a href="http://localhost:8080/couch/getFile?dbName=xxx&file=test.xml">get-file</a>
```

HTTP Response using HTTP – Headers (Download-Flag, MIME Type, Encoding)

```
header('Content-Type: ' . $mimeType);
header('Content-Disposition: attachment; filename="' . $fileName . '"');
header('Content-Transfer-Encoding: binary');
```

Data Ressources - CRUD

Using HTTP (only) GET, POST, PUT, DELETE…



POST	Creates a new resource.
GET	Retrieves a resource.
PUT	Updates an existing resource.
DELETE	Deletes a resource.

http://www.restapitutorial.com/lessons/httpmethods.html

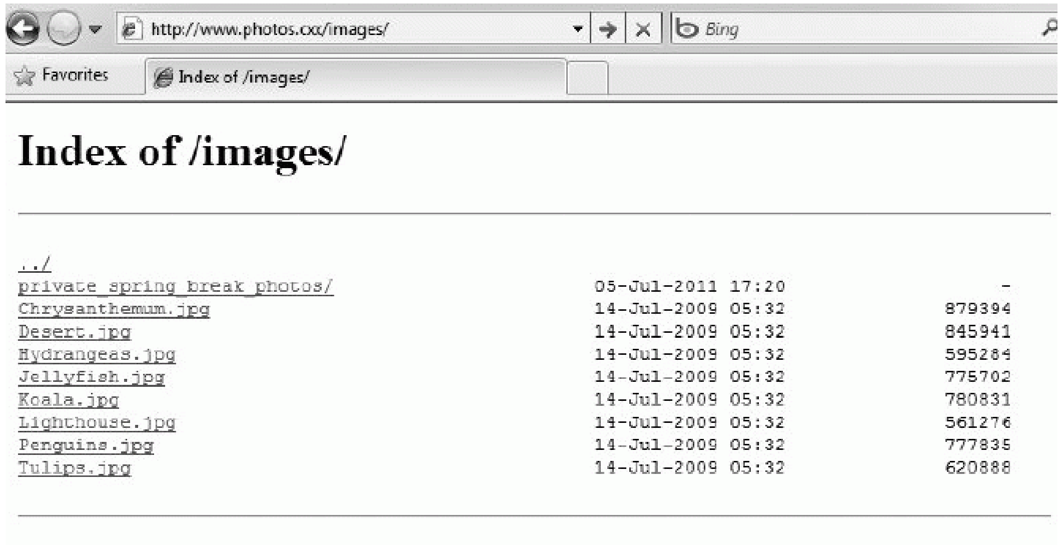
https://tools.ietf.org/html/rfc7231

Data Ressource Attacks

- **Directory Listing & File Enumeration**
- **Directory Traversal**
- **File Inclusion**
- Parameter Tampering (Form-Fields, Max-Upload-Filesize, etc.)
- …

Directory Listing & File Enumeration Attack

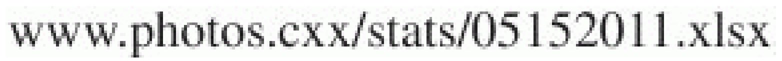
Enter a URL of a folder



Index of /images/

```
../
private spring break photos/                05-Jul-2011 17:20           -
Chrysanthemum.jpg                           14-Jul-2009 05:32      879394
Desert.jpg                                  14-Jul-2009 05:32      845941
Hydrangeas.jpg                              14-Jul-2009 05:32      595284
Jellyfish.jpg                               14-Jul-2009 05:32      775702
Koala.jpg                                   14-Jul-2009 05:32      780831
Lighthouse.jpg                              14-Jul-2009 05:32      561276
Penguins.jpg                                14-Jul-2009 05:32      777835
Tulips.jpg                                  14-Jul-2009 05:32      620888
```

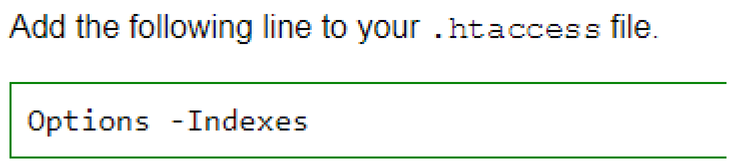
Enter a URL of a File with a timestamp or auto-increment (prevent easy guessable parameters)



www.photos.cxx/stats/05152011.xlsx

Directory Listing & File Enumeration Prevention

Prevent Listings (on Apache, per directory)

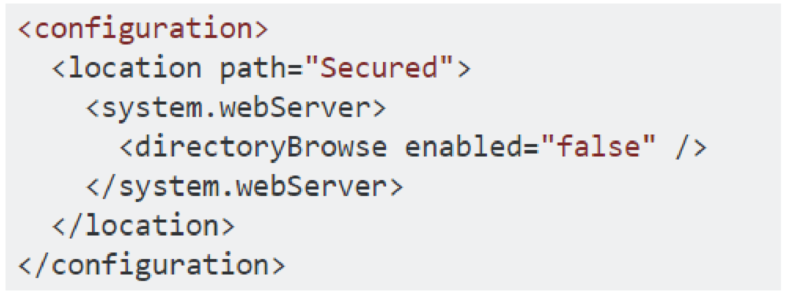


Add the following line to your `.htaccess` file.

```
Options -Indexes
```

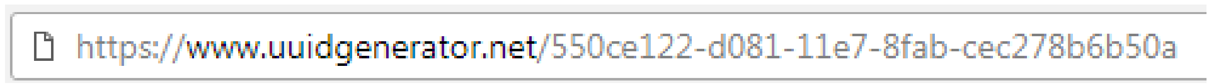
…

… on IIS



```
<configuration>
  <location path="Secured">
    <system.webServer>
      <directoryBrowse enabled="false" />
    </system.webServer>
  </location>
</configuration>
```

Prevent Enumerations: Use UIDs only



https://www.uuidgenerator.net/550ce122-d081-11e7-8fab-cec278b6b50a

Directory Traversal Attack

Consider the following Gallery-List

```
<html>
  <body>
    ...
    <a href="view_photo.php?picfile=mt_rainier.jpg">Mount Rainier
sunset</a>
    <a href="view_photo.php?picfile=space_needle.jpg">Space Needle</a>
    <a href="view_photo.php?picfile=troll.jpg">Fremont Bridge Troll</a>
  </body>
</html>
```

… and the following manual entered Link:

```
http://www.photos.cxx/view_photo.php?picfile=../private/cancun.jpg
```

Directory Traversal Prevention

```
http://www.photos.cxx/view_photo.php?picfile=../private/cancun.jpg
```

- Input Filtering
- Regex
- Use UIDs
- …

File Inclusion Attack

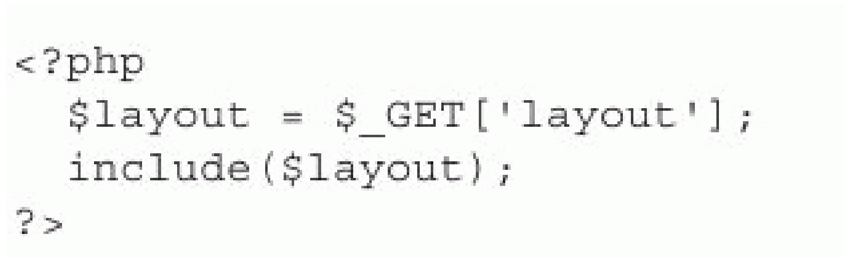
Consider the following HTML-Form

```
<html>
  <body>
    ...
    <form method="get">
      <select name="layout">
        <option value="standard.php">Standard layout</option>
        <option value="simple.php">Simple layout</option>
      </select>
      <input type="submit" />
    </form>
  </body>
</html>
```

Problem, when the GET-Parameter is directly used as a script-loader!

```
<?php
  $layout = $_GET['layout'];
  include($layout);
?>
```

File Inclusion Prevention



```
<?php
  $layout = $_GET['layout'];
  include($layout);
?>
```

- Whitelisting: Is_inArray($layout, array(«allowed_page_1», «allowed_page_2», …))
- Regex
- Filtering
- …

Lab – Databases, XSS Filter, SQL Injections, …

Idea:
Create a MVC Application where the **Login-Form** has to secured against **SQL-Injection Attacks** and **Feedback-Form** have to be secured against **XSS-Injection Attacks.**
-> See Tutorial, or:

Steps:
1. Create a MVC Application
2. Create a HTML Login Form
3. Create DB and Connect to it
4. Create User-Table
5. Secure SQL-Injection (Custom Regex and Prepared Statements) on Login Form
6. Create Feedback-Table
7. Secure Feedback-Form against XSS-Injection (using Custom Regex and Prepared Statements)

Ressources:
- https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/database-first-development/creating-the-web-application
- https://www.uuidgenerator.net
- http://rubular.com/