

# Introduction to Secure Web Programming

Apostolos Mpessas

Patras Linux User Group

December 17, 2009

```
int getRandomNumber()  
{  
    return 4; // chosen by fair dice roll.  
             // guaranteed to be random.  
}
```

<http://xkcd.com/221/>

Part I

Introduction

Secure by design



ICANHASCHEEZBURGER.COM 🐱 🐱

<http://icanhascheezburger.com>

# Levels

- Physical level
- OS level (e.g. permissions, unnecessary services, patches)
- Network level
- Web server / RDBMS
- Framework (e.g. php.ini)





Don't trust user input!

Part II

Client

# Validating User Input

- age
- credit card number
- telephone
- date
- email

# Validating User Input

- age
- credit card number
- telephone
- date
- email

# Solution

```
<html>
  <head>
    <script type="text/javascript">
      function checkPhone() {
        var phone = document.getElementsByName("phone")[0];
        var validPhone = new RegExp(/^?\d{10}$/);
        if (validPhone.test(phone.value) == true)
          return true;
      }
    </script>
  </head>
  <body>
    <form action="phone.php" method="GET">
      <input type="text" name="phone" size="10" maxlength="10"
        />
      <input type="submit" onclick="checkPhone();" />
    </form>
  </body>
</html>
```

## GET vs POST

So you hacked our site!

`http://thedailywtf.com/Articles/  
So-You-Hacked-Our-Site!.aspx`

Part III

SQL Injection

# SQL Injection

```
<?php
$query = "SELECT * FROM users WHERE user='" . $_POST['username']
        ] . "' AND password='" . $_POST['password'] . "'";
mysql_query($query);

echo $query;
?>
```

# SQL Injection

```
<?php
$query = "SELECT * FROM users WHERE user='" . $_POST['username']
        ] . "' AND password='" . $_POST['password'] . "'";
mysql_query($query);

echo $query;
?>

$_POST['username'] = 'admin';
$_POST['password'] = "' OR ''='";
```

# SQL Injection

```
<?php
$query = "SELECT * FROM users WHERE user='" . $_POST['username']
        ] . "' AND password='" . $_POST['password'] . "'";
mysql_query($query);

echo $query;
?>

$_POST['username'] = 'admin';
$_POST['password'] = "' OR ''='";

$_POST['password'] = "x'; DROP TABLE users;--";
```

## mysql\_real\_escape\_string

- ➔ Note: *If magic\_quotes\_gpc is enabled, first apply stripslashes() to the data. Using this function on data which has already been escaped will escape the data twice.*
- ➔ Note: *mysql\_real\_escape\_string() does not escape % and \_. These are wildcards in MySQL if combined with LIKE, GRANT, or REVOKE.*

*<http://gr.php.net/manual/en/function.mysql-real-escape-string.php>*

# UNION Injection

```
SELECT header, txt FROM news UNION ALL SELECT name, pass FROM members
```

# Stages of Execution

- parsing
- rewriting
- planning
- execution

In PostgreSQL

# Example

```
#!/usr/bin/env python

import sqlite3

conn = sqlite3.connect('example')

c = conn.cursor()
c.execute('''create table users (username text, password text)
''')
c.execute('insert into users values (?, ?)', ('user', 1))
conn.commit()
c.execute('select * from users where user=?', ('user', ))
c.close()
conn.close()
```

# PHP & Prepared Statements

*Note: If you are using MySQL versions 4.1.3 or later it is strongly recommended that you use the mysqli extension instead.*

*<http://gr.php.net/manual/en/mysqli.overview.php>*

# Joomla!

→ Prepared Statements? **X**

# Joomla!

- Prepared Statements? **X**
- Transactions? **X**

# ORM

- SQLAlchemy (python)
- SQLObject (python)
- Active Record (ruby)
- Doctrine (php)

# ORM

- SQLAlchemy (python)
- SQLObject (python)
- Active Record (ruby)
- Doctrine (php)

# Roles & Privileges

RDBMs support roles and privileges.

# Roles & Privileges

RDBMs support roles and privileges. Use them!

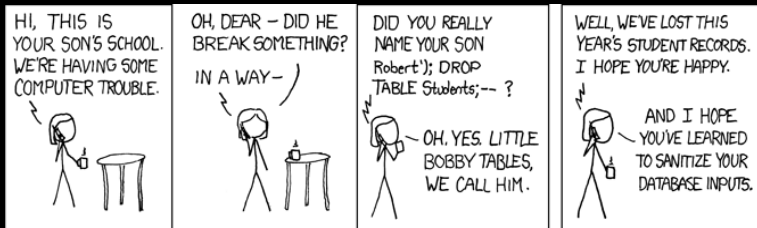
# Roles & Privileges

RDBMs support roles and privileges. [Use them!](#)

- table access
- operations allowed (INSERT, UPDATE, DELETE, SELECT)
- column/row level permissions

ENCRYPT!

<http://xkcd.com/327/>



Part IV

XSS

# Same Origin Policy

- Javascript
- HTTP cookies

# XSS

*Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications which allow code injection by malicious web users into the web pages viewed by other users.*

*[http://en.wikipedia.org/wiki/Cross-site\\_scripting](http://en.wikipedia.org/wiki/Cross-site_scripting)*

# Types

- Non-persistent (reflected)
- Persistent (stored)

```
<div id="signature">  
  <p><?php echo $signature ?></p>  
</div>
```

```
<div id="signature">
  <p><?php echo $signature ?></p>
</div>
```

```
$signature = "Hi!<script type='text/javascript'>document.cookie
.do()</script>";
```

```
<a href="<?php echo $url ?>">blog</a>
```

```
<a href="<?php echo $url ?>">blog</a>
```

```
$url = "http://legit.url\" onclick="document.cookie.steal()" "  
title=\"";
```

# HTML Special Characters

Character	Representation
&	&amp;
"	&quot;
'	&#039;
<	&lt;
>	&gt;

Just replace them?

```

```

Just replace them?

```

```

Just replace them?

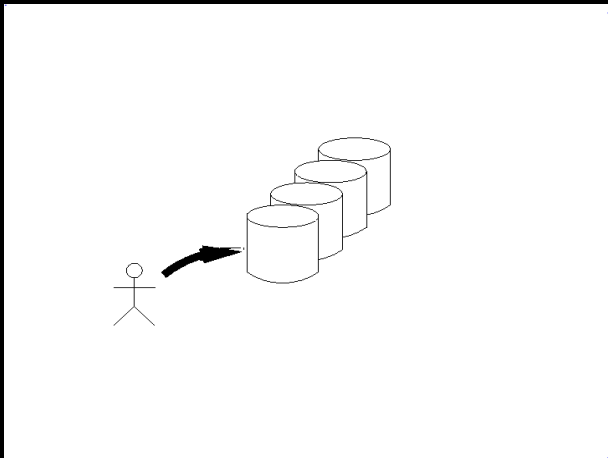
```

```

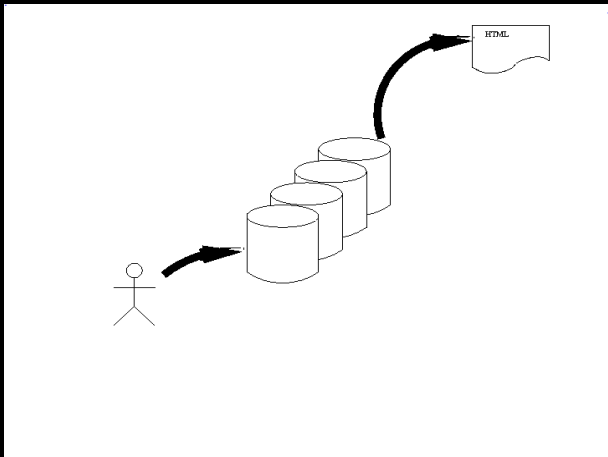
# PHP functions

- `htmlspecialchars()`
- `strip_tags()`

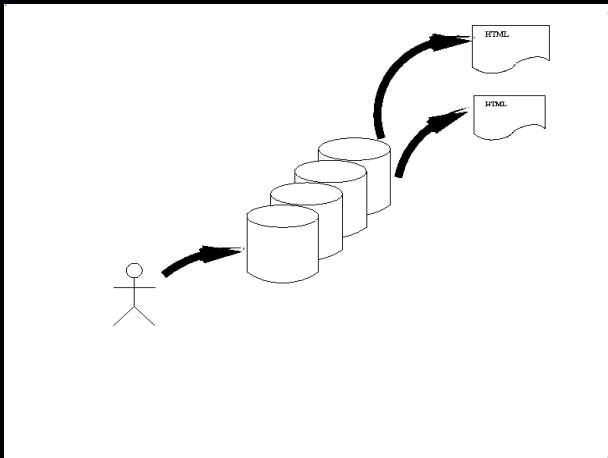
# Input Validation vs Output Sanitization



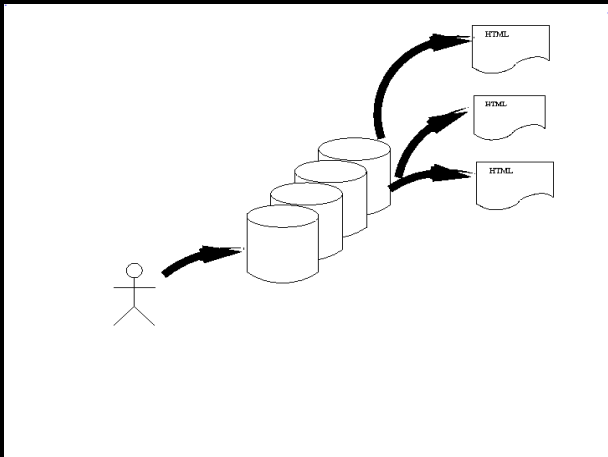
# Input Validation vs Output Sanitization



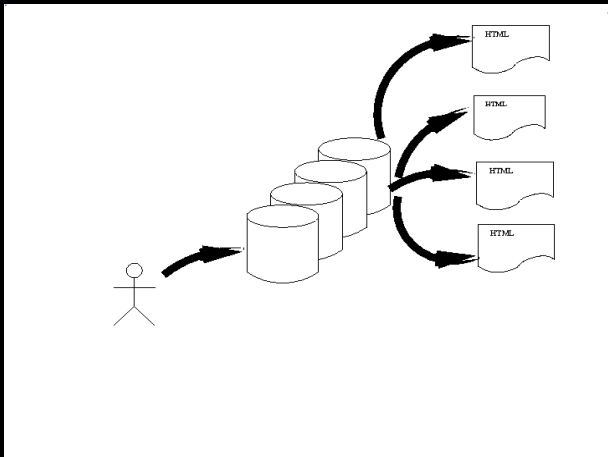
# Input Validation vs Output Sanitization



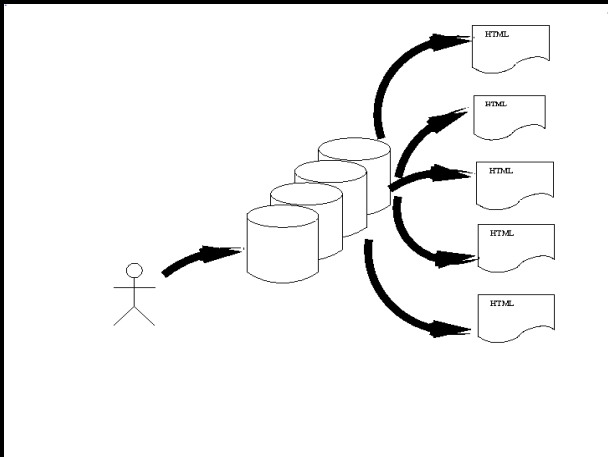
# Input Validation vs Output Sanitization



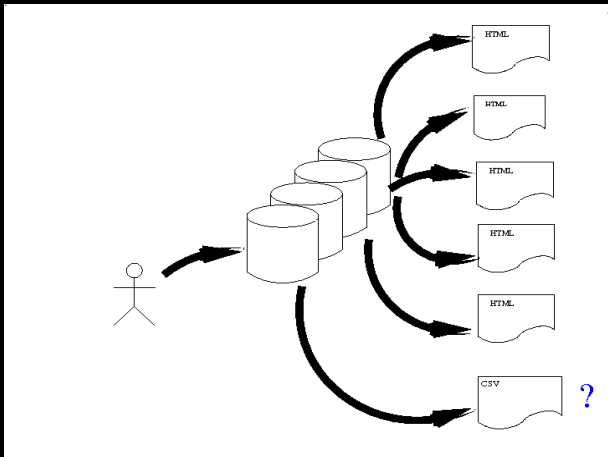
# Input Validation vs Output Sanitization



# Input Validation vs Output Sanitization



# Input Validation vs Output Sanitization



Part V

CSRF

# CSRF

*Cross-site request forgery, also known as a one-click attack or session riding and abbreviated as CSRF (“sea-surf”) or XSRF, is a type of malicious exploit of a website whereby unauthorized commands are transmitted from a user that the website trusts.*

*<http://en.wikipedia.org/wiki/CSRF>*

# Examples

→ ``

# Examples

→ ``

→ `<div style="display:none">  
 <form action="http://example.com/delete.php" method="POST">  
 <input type="hidden" name="id" value="1" />  
 </form>  
</div>`

`<script>document.forms[0].submit()</script>`

## Solution

- Differentiate requests made by the *user* for *your* website from requests made by his *browser* for a *different* website.
- Use form tokens (**crumbs**) as a hidden field so as to identify “legitimate” requests.
  - Unique per user
  - Should be changed often

# Login CSRF

Remember to protect login forms as well!

# The spider of doom

http:

//thedailywtf.com/Articles/The\_Spider\_of\_Doom.aspx

Part VI

Clickjacking

# Clickjacking

*Clickjacking is a malicious technique of tricking Web users into revealing confidential information or taking control of their computer while clicking on seemingly innocuous Web pages. A vulnerability across a variety of browsers and platforms, a clickjacking takes the form of embedded code or script that can execute without the user's knowledge, such as clicking on a button that appears to perform another function.*

*<http://en.wikipedia.org/wiki/Clickjacking>*

## How it works

→ iframe with zero opacity

# Framekiller

```
<script type="text/javascript">
  if (top != self) {
    top.location.replace(self.location.href);
    //alert('busting you out, please wait...');
  }
</script>
```

# Framekiller

```
<script type="text/javascript">  
  if (top != self) {  
    top.location.replace(self.location.href);  
    //alert('busting you out, please wait...');  
  }  
</script>
```

→ User needs Javascript *enabled* for this!

# Framekiller

```
<script type="text/javascript">
  if (top != self) {
    top.location.replace(self.location.href);
    //alert('busting you out, please wait...');
  }
</script>
```

- User needs Javascript *enabled* for this!
- <http://en.wikipedia.org/wiki/Framekiller>

# Framekiller

```
<script type="text/javascript">
  if (top != self) {
    top.location.replace(self.location.href);
    //alert('busting you out, please wait...');
  }
</script>
```

- User needs Javascript *enabled* for this!
- <http://en.wikipedia.org/wiki/Framekiller>
- NoScript Add-On for Firefox

Part VII

Cookies

# PHP Sessions

- Use the filesystem
- Use the memory
- Use a custom handler (e.g. database)

Same Origin Policy

# Signed Cookies

Cryptographically sign the cookie.

Part VIII

Epilogue

## Secure At Last!

*More than 70% of people would reveal their computer password in exchange for a bar of chocolate, a survey has found.*

*It also showed that 34% of respondents volunteered their password when asked without even needing to be bribed.*

*<http://news.bbc.co.uk/2/hi/technology/3639679.stm>*

# Bibliography



Dowd Mark, McDonald John, Schuh Justin

*The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*

Addison-Wesley Professional, November 2006



Daswani Neil, Kern Christoph, Kesavan Anita

*Foundation of Security: What Every Programmer Needs to Know*

Apress, 2007



Simon Willison

*Web Security Horror Stories*

<http://simonwillison.net/2008/talks/head-horror/>