Objectives

- Raise awareness of security issues in PHP
- Manage site logins
CS Department Policy

The following information is never to be used with malicious intent, or to “show off”. It is understood that to write secure code, one must comprehend what makes code insecure and how or why it is insecure.

Use of techniques discussed in class without prior approval of all parties involved will result in termination from the CS department, and possible discipline measures from the university and/or local authorities.

Security

Security is an implicit requirement in every web application involving server-side scripting. Although covered "last", security should be a primary design consideration.

Content about security in this presentation draws heavily from Essential PHP Security by Chris Shiflett, and it is a highly recommended resource.
Identifying the Threats

Four types of threats to server side applications

- User permissions (who sees what)
- What to store, what not to store
- Encoding data sent to server using SSL
- Deleting a table
- Loss of a server due to a destructive event, e.g., natural disaster

Identifying the Threats (continued)

- Crashing the computer
- Filling up storage
- Generating multiple processes, using up memory
- Causing hardware failure on server by manipulating device drivers
- Flooding network with traffic
- SQL Injection
- Cross Site Scripting (XSS)
Important security concerns

Never trust data coming from the user. It could be

Do not turn "keys to site content" over to users.
Always validate form input. Consider that it may be possible for user to create own form and feed it to your script.

Never use hidden form elements for anything truly important.

Caching

Many web browsers will cache data. We do not want that to happen with our dynamic content.
Most browsers will disable caching with the following:

```php
<?php
header("Cache-Control: no-cache, must-revalidate");
// HTTP/1.1
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
// Date in the past
?>
```
SQL Injection

Exploiting an application that takes data from user input and uses it to form
an SQL query without proper "sanitation".

Example: form asks for username and password. Processing script uses
that to build query:

Select * from members_tbl where username = 'xx' and password = 'yy

Instead of entering a real username and password, user enters the
following:

For username: ' or ''='' (sq or sq sq= sq)
For password: ' or "=" (sq or sq sq = sq)

Producing the following query:

Select * from members_tbl where username = " or "=" and password
= " or "="

The above query is ... where username = " or TRUE and password = " or
TRUE
Preventing

Every time you give user chance to enter data, you MUST check to be sure not trying to manipulate your application.

Create and use a `clean()` function

```php
function clean($input, $maxlength)
{
    $input = trim($input);
    $input = substr($input, 0, $maxlength);
    $input = escapeshellcmd($input);
    $input = htmlspecialchars($input, ENT_QUOTES);
    return $input;
}
```

`escapeshellcmd()` escapes characters that might be used to trick a shell command into executing arbitrary commands.

`htmlspecialchars()` prevents user-supplied text from containing HTML markup.

Cross site Scripting

Embedding in content passed to script (and displayed without cleaning) a client-side script.

Scenario:

Malicious visitor visits our guest book page and instead of their name supplies the following "<script>alert(document.cookie)</script>"

This script will now be sent to every site visitor's browser.

The example just causes a dialog box to pop up showing the targeted site's cookies on the user's computer. This technique could be used, however, to send cookies to another site for collection.
Providing site login

Mechanics of coding are pretty straightforward based on what we know at this point,

however

Big challenge is to think through the entire system and make sure every logistical aspect of user registration (and related maintenance) is considered.
## Supporting Site Login

### Method of establishing account
- Will account be strictly "web based"?
- Will it be tied to an established customer account in another system? (banking, investing, etc.) How do we make the connection?

### Names used on system
- Are any names fine?
- Will people be able to select a pseudonym?
- Will people login with their user name or email address?

### What happens if someone forgets username and/or password?
- Will account be "tied" to an email address for verification?
  - If not, how do we establish "control"?
  - If so, what happens if someone changes their email address?

### Should our site support anonymity or not? What's our goal?
- Site providing account information for credit card company vs. message board to discuss sports
**Mechanics of logging in**

Common to allow user to login via link off the main page.

Once logged in, do we make use of cookies so user does not have to specifically log in again?

- How does this affect overall security?
- Not wise to store this in clear text in a cookie on user's machine.

Regardless of above, we'll need to keep login persistent for this user session.

- Will user be logged off after a period of inactivity?

---

**Persistence of login**

Once user has logged in, establish a session variable denoting their login and also storing such info as their user name, etc.

If it is necessary to update a user profile from a database on an ongoing basis, then it may be better to store username and password as session variables and login to database on each page.

- Example...message board that keeps track of number of user posts.

User should be permitted to logout by choice.

Can user change username? password? other account attributes (address, etc.)?
### Login mechanics

Are we having user login simply to validate their identity, or are we tying that identity to other things.

Example: Suppose user login entitles them to look at bank records for a particular account. How do we establish relationship between bank transactions table and user's identity?

Do we care if user somehow logs in simultaneously from more than 1 machine?

Does user's IP address matter?

---

### Example

Design a database with a table containing user name and password.

User name should be primary key. *(why?)*

For security sake, don't store password in plain text.

Use either crypt or MD5.

- crypt uses DES to encrypt only the first 8 characters of a message and returns it as an 8 character string.
- md5 uses an RSA encryption algorithm to create a 32 character 'message digest' of a message and returns it as a 32 character string.

Both of the above are 1 way (encryption) only.
Example

Create an authenticate_user function to take in a username and password and check to see if in table.
If so, set a cookie or environmental variable or just return true.
Call this function (perhaps through use of "include" on any secure page).