# PHP Forms

Class 10

## Danger!

- there is a huge security hole in processform.php
- it is vulnerable to an injection attack

**Injection Attack**

The execution or interpretation of malicious data (code) in an unexpected manner.

- an attack is always possible when external data is supplied
- here we are accepting input from a user and using it without validation in the PHP program

Example with reportform $\longrightarrow$ processform

## Defense: Sanitize Input

change

```
$what_they_did = $_POST['whattheydid'];
```

into

```
$what_they_did = htmlspecialchars($_POST['whattheydid']);
```

for every `$_POST` variable

note that this simple solution only works for pure HTML controls in a PHP program

more complex scenarios (e.g., JavaScript) require more sophisticated security measures

## Data Leak

- there is a second huge security hole in this system `https://borax.truman.edu/315/c09/results.txt`
- the file has to be world-readable
- so anyone can see the contents

- solution: move the file to a place where the webserver cannot see it
- PHP is a program running on the server
- it can access the entire filesystem, not just the web hierarchy

## POST vs GET

there are two main request types a browser can send to a server

- a GET request
- a POST request

there are a number of others, but these are the big two

## GET

- a GET request transmits information to the server in the URL
- this is the default
- `http://borax.truman.edu/315/index.html` is a GET request
- there is no additional information

- the URL can be expanded to send additional information
- for example:
  `http://borax.truman.edu/315/index.html?foo=bar`

- this example does nothing, because a plain HTML page can't respond to GET parameters

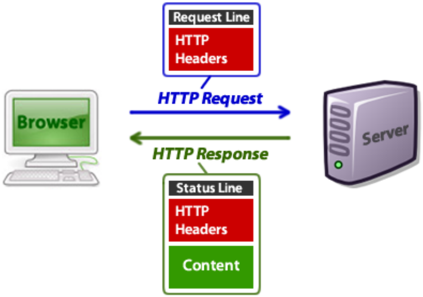
## GET Parameters

- however, a PHP program can respond to GET parameters
- `http://borax.truman.edu/315/c10/echo.php`
- `http://borax.truman.edu/315/c10/echo.php?foo=bar&bim=bam`
- this GET request sends information to a PHP program that can accept its values and do something with them

## POST and HTTP Headers

- a POST request also sends information to the server
- it does so in the HTTP headers
  (this is totally different from HTML header stuff)

- you don't see them, but every HTTP exchange involves headers
- you think you just sent a URL request, but you sent headers also



## GET Parameters

GET parameters (name and value) are

- embedded in the URL — literally typed in the URL

```
http://borax.truman.edu/315/c10/echo.php?foo=bar
```

## POST Parameters

POST parameters (name and value) are

- embedded in the headers
- only occur if the page has a form
- result from input, select, and textarea elements
- are sent when the form is submitted

this is how processform.php works

## POST Values

- watch what happens when we submit reportform.html to a program that echoes
- `http://borax.truman.edu/315/c10/reportform_echo.html`

## Pros and Cons

GET Pros

- simple — parameters and values are visible
- can be bookmarked or sent via email — the entire request is in the URL

POST Pros

- unlimited data size — not limited to 150 bytes
- parameters and values are secure because they're hidden

## GET and POST

Classic scenarios

- GET
    - idempotent requests
    - requesting information that requires parameters
        - executing a search
        - requesting a page of photos from a gallery
        - requesting an article
- POST
    - requests that cause changes to server state
    - submitting new data to be saved on the server
    - submitting sensitive or private parameters or data

## Quiz

http://borax.truman.edu/315/c10/quiz.php

- the web is stateless, but with PHP we can trick it into being stateful
- `<input type="hidden"` is a very simple way

## Syntax Errors

- a syntax error will halt program execution immediately, or prevent it entirely
- almost always this results in a blank page
- this is simple to diagnose: run the program from the command line:

```
$ php -f syntax.php

PHP Parse error:  syntax error, unexpected '$lines'
(T_VARIABLE), expecting '{' in
/home/jbeck/crs/221/315/c10morephp/syntax.php on line 31
```

## Warnings

- warnings will not halt program execution, but are logical errors
    - strategy 1: run from command line as above
    - strategy 2: include the following code at the top of the PHP program:

```
error_reporting(E_ALL);
ini_set('display_errors', '1');
```

- note this is only during development, not in production
- on production system, can leak details that hackers can use

## Incorrect Behavior

- no syntax errors or warnings, but unexpected results
- strategy: debugging output

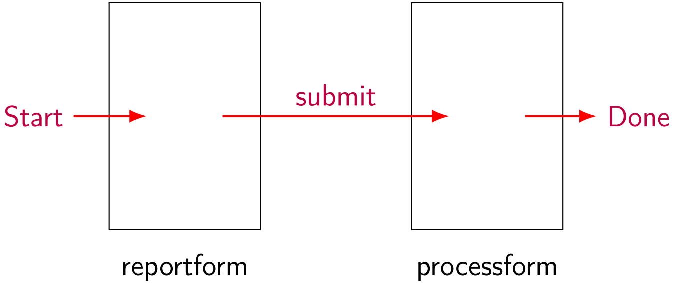
```
<pre><?php print_r($_POST); ?>
</pre>
```

- or `var_dump($foo);` which is more verbose

- must do this after headers are emitted

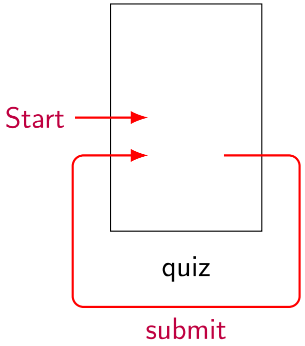
## A Sequence of Actions

- the alien abduction form tag in reportform.html
  `<form method="post" action="processform.php">`



## A Sequence of Actions

- the quiz form tag in quiz.php
  `<form method="post" action="quiz.php">`



## A Sequence

- file1 $\rightarrow$ file2 $\rightarrow$ $\cdots$
  - pro: easy to understand
  - pro: each file represents a state in the sequence
  - con: if a user manually types a URL in the middle of the sequence, everything's confused
- submitting to yourself
  - pro: user can't jump into the middle
  - con: have to determine where in the sequence you are; easy to get confused