CSC 405
Computer Security

Web Security

Alexandros Kapravelos
akaprav@ncsu.edu

(Derived from slides by Giovanni Vigna and Adam Doupe)

NC STATE UNIVERSITY

https://airma.sh

CristianRodriguez
Destroyed by Dr.HelenaRazo
Dr.HelenaRazo

How would you defend against this attack?

http://go.ncsu.edu/airmash

HTML – Forms

- A `form` is a component of a Web page that has form controls, such as text fields, buttons, checkboxes, range controls, or color pickers
  - Form is a way to create a complicated HTTP request
- `action` attribute contains the URI to submit the HTTP request
  - Default is the current URI
- `method` attribute is the HTTP method to use in the request
  - GET or POST, default is GET

HTML – Forms

- Children `input` tags of the form are transformed into either query URL parameters or HTTP request body
- Difference is based on the `method` attribute
  - GET passes data in the query
  - POST passes data in the body
- Data is encoded as either “application/x-www-form-urlencoded” or “multipart/form-data”
  - GET always uses “application/x-www-form-urlencoded”
  - POST depends on `enctype` attribute of `form`, default is “application/x-www-form-urlencoded”
  - "multipart/form-data" is mainly used to upload files, so we will focus on “application/x-www-form-urlencoded”

HTML – Forms

- Data sent as name-value pairs
  - Data from the input tags (as well as others)
    ```
    <input type="text" name="foo"
    value="bar">
    ```

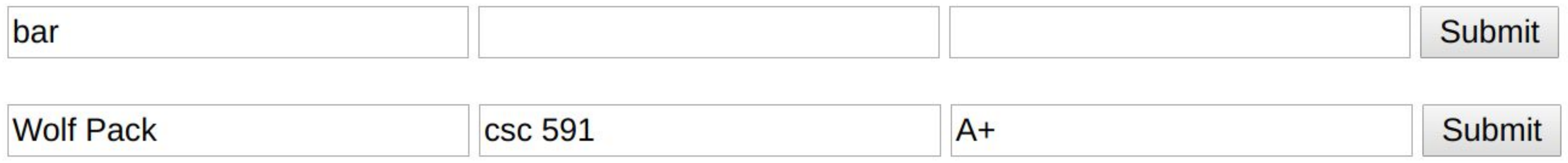
- Name is taken from the `input` tag's `name` attribute
- Value is taken either from the `input` tag's `value` attribute or the user-supplied input
  - Empty string if neither is present

application/x-www-form-urlencoded

- All name-value pairs of the form are encoded
- form-urlencoding encodes the name-value pairs using percent encoding
  - Except that spaces are translated to + instead of %20
- `foo=bar`
- Multiple name-value pairs separated by ampersand (&)

application/x-www-form-urlencoded

```
<form action="http://example.com/grades/submit">
  <input type="text" name="student" value="bar">
  <input type="text" name="class">
  <input type="text" name="grade">
  <input type="submit" name="submit">
</form>
```

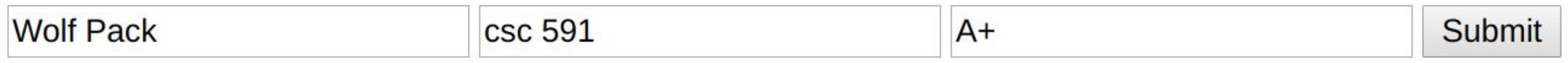


bar			Submit
Wolf Pack	csc 591	A+	Submit

```
http://example.com/grades/submit?student=Wolf+Pack&
class=csc+591&grade=A%2B&submit=Submit
```

application/x-www-form-urlencoded

```
<form action="http://example.com/grades/submit" method="POST">
  <input type="text" name="student" value="bar">
  <input type="text" name="class">
  <input type="text" name="grade">
  <input type="submit" name="submit">
</form>
```



Wolf Pack	csc 591	A+	Submit

```
POST /grades/submit HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:34.0) Gecko/20100101 Firefox/34.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 68

student=Wolf+Pack&class=csc+591&grade=A%2B&submit=Submit
```

Web Applications

- It was quickly realized that the way the web was structured allowed for returning dynamic responses
- Early web was intentionally designed this way, to allow organizations to offer access to a database via the web
- Basis of GET and POST also confirm this
  - GET "SHOULD NOT have the significance of taking an action other than retrieval"
    - Safe and idempotent
  - POST
    - Annotation of existing resources; posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles, providing a block of data, such as the result of submitting a form, to a data-handling process; and extending a database through an append operation

Web Applications

- Server-side code to dynamically create an HTML response
- How does this differ from a web site?
- In the HTTP protocol we've looked at so far, each request is distinct
  - Server has client IP address and User-Agent

Maintaining State

- HTTP is a stateless protocol
- However, to write a web application we would like maintain state and link requests together
- The goal is to create a "session" so that the web application can link requests to the same user
  - Allows authentication
  - Rich, full applications
- Three ways this can be achieved
  - Embedding information in URLs
  - Using hidden fields in forms
  - Using cookies

Embedding Information in Cookies

- Cookies are state information that is passed between a web server and a user agent
  - Server initiates the start of a session by asking the user agent to store a cookie
  - Server or user agent can terminate the session
- Cookies first defined by Netscape while attempting to create an ecommerce application
- RFC 2109 (February 1997) describes first standardization attempt for cookies
- RFC 2965 (October 2000) tried to standardize cookies 2.0
- RFC 6265 (April 2011) describes the actual use of cookies in the modern web and is the best reference

Embedding Information in Cookies

- Cookies are name-value pairs (separated by "=")
- Server includes the "Set-Cookie" header field in an HTTP response
  - `Set-Cookie: USER=foo;`
- User agent will then send the cookie back to the server using the "Cookie" header on further requests to the server
  - `Cookie: USER=foo;`

Embedding Information in Cookies

- Server can ask for multiple cookies to be stored on the client, using multiple "Set-Cookie" headers
  - `Set-Cookie: USER=foo;`
  - `Set-Cookie: lang=en-us;`

Embedding Information in Cookies

- Server can sent several attributes on the cookie, these attributes are included in the Set-Cookie header line, after the cookie itself, separated by `";"`
  - `Path`
    - Specifies the path of the URI of the web server that the cookies are valid
  - `Domain`
    - Specifies the subdomains that the cookie is valid
  - `Expires` or `Max-Age`
    - Used to define the lifetime of the cookie, or how long the cookie should be valid
  - HttpOnly
    - Specifies that the cookie should not be accessible to client-side scripts
  - Secure
    - Specifies that the cookie should only be sent over secure connections

Embedding Information in Cookies

- Example cookie headers from curl request to www.google.com
  - curl -v http://www.google.com
- `Set-Cookie: PREF=ID=db9539b9b7353be5:FF=0:TM=1421424672:LM=1421424672:S=OqGXMZZhmeyihyKi; expires=Sun, 15-Jan-2020 16:11:12 GMT; path=/; domain=.google.com`
- `Set-Cookie: NID=67=bs1lLyrXtfdUj79IlcuqR7_MWEsyNdLWU_FpGKwlWR9QpEzi3UrVV2UGO6LBW3sJNk9mlLcYIJns3PG3NUu-M3pT9qD-V4F8oyyJ_UJnCGKDUDGbllL9Ha8KGufv0MUv; expires=Sat, 18-Jul-2020 16:11:12 GMT; path=/; domain=.google.com; HttpOnly`

- `Set-Cookie: PREF=ID=db9539b9b7353be5:FF=0:TM=1421424672:LM=1421424672:S=OqGXMZZhmeyihyKi; expires=Sun, 15-Oct-2019 16:11:12 GMT; path=/; domain=.google.com`
  - `expires` is set two years in the future
  - `path` is / which means to send this cookie to all subpaths of www.google.com/
  - `domain` is .google.com, which means to send this cookie to all subdomains of .google.com
    - Includes www.google.com, drive.google.com, …

- Set-Cookie: NID=67=bs1lLyrXtfdUj79IlcuqR7_MWEsyNdLWU_FpGKwlWR9QpEzi3UrVV2UGO6LBW3sJNk9mlLcYIJns3PG3NUu-M3pT9qD-V4F8oyyJ_UJnCGKDUDGbllL9Ha8KGufv0MUv; expires=Sat, 18-Jul-2015 16:11:12 GMT; path=/; domain=.google.com; $\boxed{\text{HttpOnly}}$
  - HttpOnly is a security feature, which means only send this cookie in HTTP, do not allow JavaScript code to access the cookie

Embedding Information in Cookies

- The server can request the deletion of cookies by setting the "expires" cookie attribute to a date in the past
- User agent should then delete cookie with that name
- `Set-Cookie: USER=foo; expires=Thu, 15-Jan-2020 16:11:12 GMT;`
    - User agent will then delete the cookie with name "USER" that is associated with this domain
- – Proxies are not supposed to cache cookie headers
    - Why?

Embedding Information in Cookies

- User agent is responsible for following the server's policies
  - Expiring cookies
  - Restricting cookies to the proper domains and paths
- However, user agent is free to delete cookies at any time
  - Space/storage restrictions
  - User decides to clear the cookies

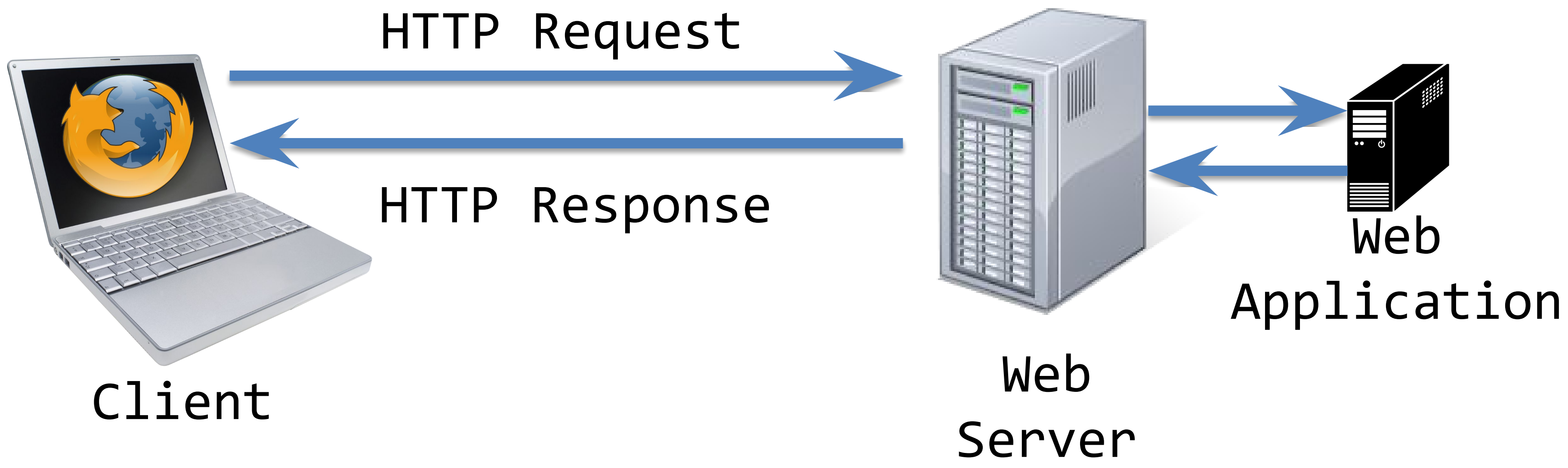
Modern Sessions

- Sessions are used to represent a time-limited interaction of a user with a web server
- There is no concept of a "session" at the HTTP level, and therefore it has to be implemented at the web application level
  - Using cookies
  - Using URL parameters
  - Using hidden form fields
- In the most common use of sessions, the server generates a unique (random and unguessable) session ID and sends it to the user agent as a cookie
- On subsequent requests, user agent sends the session ID to the server, and the server uses the session ID to index the server's session information

Designing Web Applications

- In the early days of the web, one would write a "web application" by writing a custom web server that received HTTP requests, ran custom code based on the URL path and query data, and returned a dynamically created HTML page
  - The drawback here is that one would have to keep the web server up-to-date with the latest HTTP changes (HTTP/1.1 spec is 175 pages)
- Generally decided that it was a good idea to separate the concerns into a web server, which accepted HTTP request and forwarded relevant requests to a web application
  - Could develop a web application without worrying about HTTP

Web Application Overview



Common Gateway Interface (CGI)

- standard protocol for web servers to execute programs
- request comes in
- web server executes CGI script
- script generates HTML output
- often under `cgi-bin/` directory
- environmental variables are used to pass information to the script
  - `PATH_INFO`
  - `QUERY_STRING`

Active Server Pages (ASP)

- Microsoft's answer to CGI scripts
- First version released in 1996
- Syntax of a program is a mix of
  - Text
  - HTML Tags
  - Scripting directives (VBScript Jscript)
  - Server-side includes (#include, like C)
- Scripting directives are interpreted and executed at runtime
- Will be supported "a minimum of 10 years from the Windows 8 release date"
  - October 26th, 2022

ASP Example

```
<% strName = Request.Querystring("Name")
   If strName <> "" Then %>
<b>Welcome!</b>
<% Response.Write(strName)
   Else %>
<b>You didn't provide a name...</b>
<% End If %>
```

Web Application Frameworks

- As the previous Request.Querystring example shows, frameworks were quickly created to assist web developers in making web applications
- Frameworks can help
  - Ease extracting input to the web application (query parameters, form parameters)
  - Setting/reading cookies
  - Sessions
  - Security
  - Database

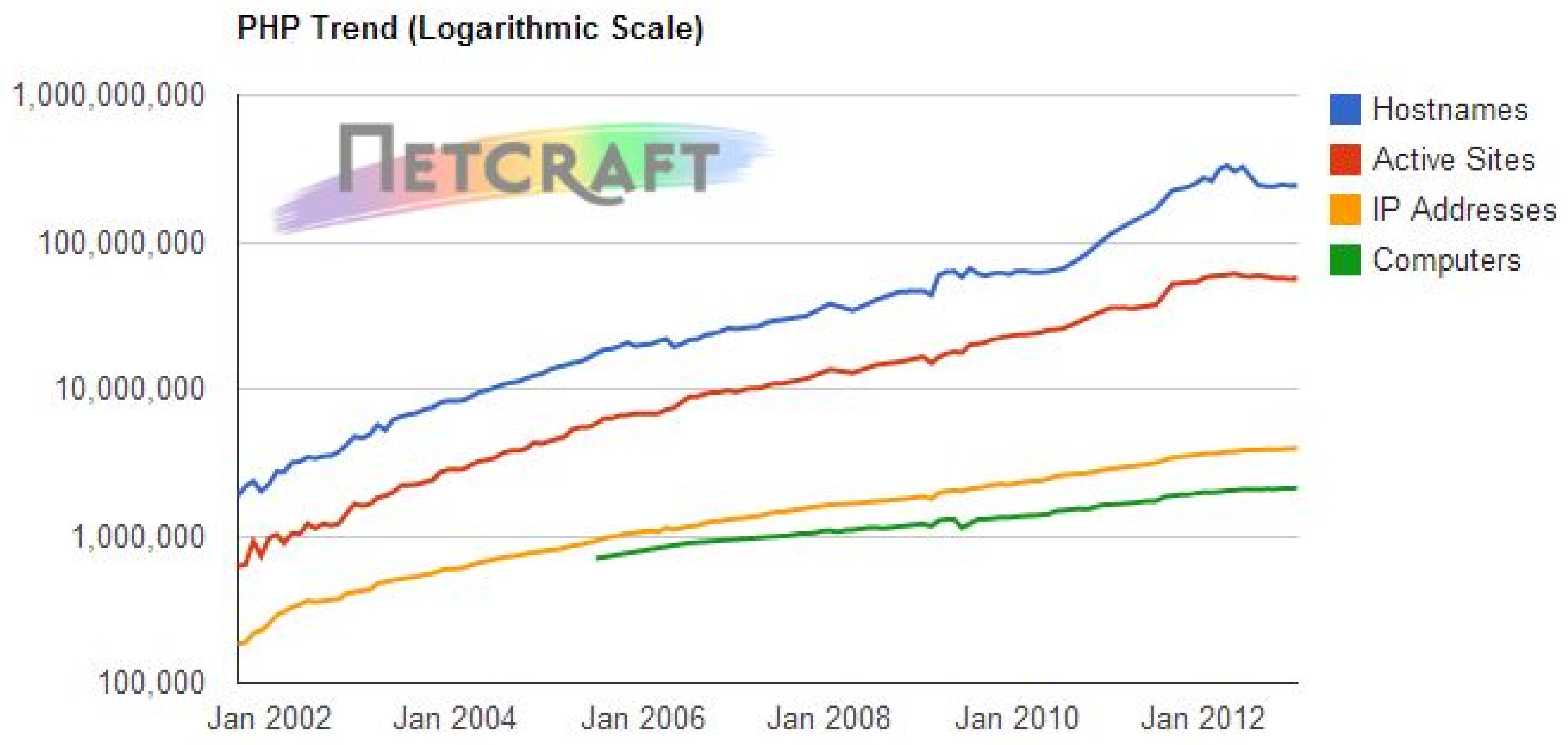
Web Application Frameworks

- Important to study web application frameworks to understand the (security) pros and cons of each
- Some vulnerability classes are only present in certain frameworks

PHP: Hypertext Preprocessor

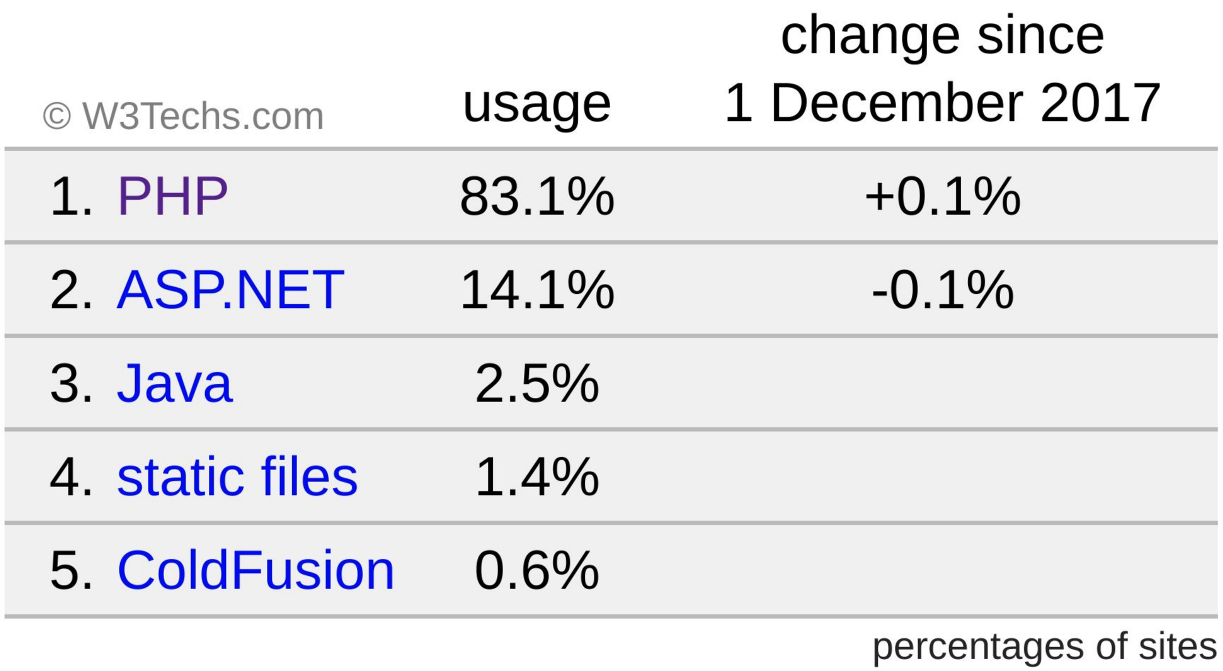
- Scripting language that can be embedded in HTML pages to generate dynamic content
  - Basic idea is similar to JSP and ASP
- Originally released in 1995 as a series of CGI scripts as C binaries
- PHP 3.0 released June 1998 is the closest to current PHP
  - "At its peak, PHP 3.0 was installed on approximately 10% of the web servers on the Internet" - http://php.net/manual/en/history.php.php
- PHP 4.0 released May 2000
- PHP 5.0 released July 2004
  - Added support for objects
- PHP 5.6 released August 2014 (still supported)
- PHP 7.2 released in November 2017 is the latest version

PHP – Popularity



http://news.netcraft.com/archives/2013/01/31/php-just-grows-grows.html

Most popular server-side programming languages



© W3Techs.com	usage	change since 1 December 2017
1. PHP	83.1%	+0.1%
2. ASP.NET	14.1%	-0.1%
3. Java	2.5%	
4. static files	1.4%	
5. ColdFusion	0.6%	

percentages of sites

source: https://w3techs.com/

PHP

- The page is parsed and interpreted on each page request
  - Can be run as CGI, so that a new copy of the PHP interpreter is run on each request
  - Or the PHP interpreter can be embedded into the web server
    - mod_php for apache
- Completely new language
  - C-like in syntax
  - Custom designed to build web applications
  - Language grew organically over time

PHP – Example

```
<!DOCTYPE html>
<html>
    <head>
        <title>PHP Hello World</title>
    </head>
    <body>
        <?php echo '<p>Hello World</p>'; ?>
    </body>
</html>
```

PHP – Features

- Dynamically typed
- String variable substitution
- Dynamic `include`/`require`
- Superglobals
- Variable variables
- `register_globals`

PHP – String Variable Substitution

```php
<?php
echo 'this is a simple string';
echo 'Variables do not $expand $either';

$juice = "apple";
echo "He drank some $juice juice.";

$juices = array("apple", "orange", "koolaid1" => "purple");
echo "He drank some $juices[0] juice.";
echo "He drank some $juices[1] juice.";
echo "He drank some $juices[koolaid1] juice.";

echo "This works: {$juices['koolaid1']}";
```

http://php.net/manual/en/language.types.string.php

PHP – Dynamic `include/require`

```
<?php
/**
* Front to the WordPress application. This file doesn't do anything, but loads
* wp-blog-header.php which does and tells WordPress to load the theme.
*
* @package WordPress
*/

/**
* Tells WordPress to load the WordPress theme and output it.
*
* @var bool
*/
define('WP_USE_THEMES', true);

/** Loads the WordPress Environment and Template */
require( dirname( __FILE__ ) . '/wp-blog-header.php' );
```

wp-blog-header.php

```
<?php
/**
* Loads the WordPress environment and template.
*
* @package WordPress
*/

if ( !isset($wp_did_header) ) {

    $wp_did_header = true;

    require_once( dirname(__FILE__) . '/wp-load.php' );

    wp();

    require_once( ABSPATH . WPINC . '/template-loader.php' );

}
```

`allow_url_include`

- PHP setting to allow http and ftp urls to include functions
- Must enable `allow_url_fopen` as well
  - This setting allows calling `fopen` on a url
- Remote file is fetched, parsed, and executed

PHP - Superglobals

```php
<?php
if ( 'POST' != $_SERVER['REQUEST_METHOD'] ) {
    header('Allow: POST');
    header('HTTP/1.1 405 Method Not Allowed');
    header('Content-Type: text/plain');
    exit;
}
$comment_post_ID = isset($_POST['comment_post_ID']) ? (int) $_POST['comment_post_ID'] : 0;

$post = get_post($comment_post_ID);
if ( empty( $post->comment_status ) ) {
    /**
      * Fires when a comment is attempted on a post that does not exist.
      * @since 1.5.0
      * @param int $comment_post_ID Post ID.
      */
    do_action( 'comment_id_not_found', $comment_post_ID );
    exit;
}
// get_post_status() will get the parent status for attachments.
$status = get_post_status($post);
$status_obj = get_post_status_object($status);
```

Wordpress – wp-comments-post.php

PHP – Variable Variables

```
<?php
$a = 'hello';
$$a = 'world';

echo "$a $hello";
echo "$a ${$a}";
```

http://php.net/manual/en/language.variables.variable.php

PHP – `register_globals`

- "To register the EGPCS (Environment, GET, POST, Cookie, Server) variables as global variables."
- PHP will automatically inject variables into your script based on input from the HTTP request
  - HTTP request variable name is the PHP variable name and the value is the PHP variable's value
- Default enabled until 4.2.0 (April 2002)
- Deprecated as of PHP 5.3.0
- Removed as of PHP 5.4.0

PHP – register_globals

```
<html>
 <head> <title>Feedback Page</title></head>
 <body>
   <h1>Feedback Page</h1>
   <?php
     if ($name && $comment) {
       $file = fopen("user_feedback", "a");
       fwrite($file, "$name:$comment\n");
       fclose($file);
       echo "Feedback submitted\n";
     }
   ?>
   <form method=POST>
     <input type="text" name="name"><br>
     <input type="text" name="comment"><br>
     <input type="submit" name="submit" value="Submit">
   </form>
 </body>
</html>
```

PHP – register_globals

```php
<?php
// define $authorized = true only if user is authenticated
if (authenticated_user()) {
   $authorized = true;
}

// Because we didn't first initialize $authorized as false, this might be
// defined through register_globals, like from GET auth.php?authorized=1
// So, anyone can be seen as authenticated!
if ($authorized) {
   include "/highly/sensitive/data.php";
}
?>
```

source: http://php.net/manual/en/security.globals.php