Attacks on Web Servers
Server-side code is executed on the server. The client cannot see the code; the client can only see the *result* of the code.

- Has access to GET and POST variables specified by the user.
- Typical actions incl. accessing databases, modifying site contents based on user input, etc.
- Outputs HTML code that will be sent as response to client’s request.
Figure 7.19: Actions performed by a web server to produce dynamic content for a client user.
According to w3techs.com’s survey:

**Server-side Programming Languages**

Most popular server-side programming languages

<table>
<thead>
<tr>
<th>Language</th>
<th>Usage</th>
<th>Change since 30 September 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHP</td>
<td>78.6%</td>
<td>+0.5%</td>
</tr>
<tr>
<td>ASP.NET</td>
<td>20.5%</td>
<td>-0.5%</td>
</tr>
<tr>
<td>Java</td>
<td>4.0%</td>
<td></td>
</tr>
<tr>
<td>ColdFusion</td>
<td>1.1%</td>
<td></td>
</tr>
<tr>
<td>Perl</td>
<td>0.8%</td>
<td>-0.1%</td>
</tr>
</tbody>
</table>

percentages of sites

**Fastest growing server-side programming languages** since 30 September 2012

<table>
<thead>
<tr>
<th>Language</th>
<th>Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHP</td>
<td>75</td>
</tr>
<tr>
<td>JavaScript</td>
<td>1</td>
</tr>
</tbody>
</table>

daily number of additional sites in the top 1 million

Find more details in the [server-side language surveys](#)
PHP is an open-source server-side scripting language used primarily to create dynamic web content.

- It was initially a set of Perl scripts used to maintain his personal homepage.
- He called the scripts “Personal Home Page Tools”.
PHP is an open-source server-side scripting language used primarily to create dynamic web content.

- It was initially a set of Perl scripts used to maintain his personal homepage.
- He called the scripts “Personal Home Page Tools”.
- PHP code is embedded in a PHP or HTML file stored at a web server.
- The web server will run the file through a PHP processing module to create an output HTML file.
- This HTML file is set to the client.
<html>
<body>
<p>Your number was <?php echo $x=$_GET['number']; ?>.</p>
<p>The square of your number is <?php $y = $x * $x; echo $y; ?></p>
</body>
</html>

**Code Fragment 7.15:** A simple PHP page.

<html>
<body>
<p>Your number was 5.</p>
<p>The square of your number is 25.</p>
</body>
</html>

**Code Fragment 7.16:** The output of the above PHP page.
The include function

Sometimes, it is desirable for server-side code to execute code contained in files other than the one that is currently run. In PHP, this is done using the `include` function.

```php
<?php
    include("header.html");
    include($_GET[’page’].".php");
    include("footer.html");
?>
```

**Code Fragment 7.17:** A PHP page that uses file inclusion to incorporate an HTML header, an HTML footer, and a user-specified page.

By reusing `header.html` and `footer.html`, it will allow a website to have a more uniform look.
The danger here is the file that the user inputs. If the server fails to validate the input, an attacker can include code from a remotely hosted file which will then be executed locally. This is called a *remote file inclusion* (RFI) attack.

For example, the user might supply the following URL:


`evilcode.php` will get executed locally.
Another version is the *local file inclusion* (LFI) attack. In this case, the executed code is actually stored in the server itself. This locality may allow an attacker access to private information. For example, the user might supply the following URL:

Another version is the *local file inclusion* (LFI) attack. In this case, the executed code is actually stored in the server itself. This locality may allow an attacker access to private information. For example, the user might supply the following URL:


A technique that exploits websites with LFI vulnerability:

- An attacker discovers an LFI vulnerability on a web server for victim.com.
- He also finds a photo upload form on the same site that allows uploading of PHP scripts. (i.e., it uses the `include` function)
- He uploads a PHP web shell and executes it on the web server using the LFI vulnerability.
- From there, he uploads and compiles programs that will help him have some or full control of the server.
Hackers 'Timthumb' Their Noses At Vulnerability To Compromise 1.2 Million Sites

WordPress plug-in vulnerability could be used to steal database content

Nov 02, 2011 | 05:08 PM | 0 Comments

By Ericka Chickowski, Contributing Editor
A vulnerability in an obscure WordPress add-on script that was discovered in August is currently being used to compromise more than 1.2 million websites — and could be easily used to siphon data out of databases hosted on servers also hosting the compromised websites, security experts warned today.
The vulnerability in question comes from the timthumb.php script, a photo-resizing utility used by many third-party WordPress plug-ins that allows hackers to write whatever content they point to as long as a few restrictions are met, says Mike Geide, senior researcher at Zscaler ThreatLabZ.

"For example, the utility might have as a check that you only pass it content from YouTube, but the check that it does will only make sure YouTube exists within the URL path, so you could create your own domain, youtube.com.evil.com, and it would pass that check, and then you could pass it phpshell.php," Geide says. A recent blog post from researchers with Sucuri Security showed how they were tracking infections from the vulnerability. A Google search today found 1.2 million sites affected by the infection.
Databases and SQL Injection Attacks

A database is a system that stores information in an organized way. Many websites use databases to efficiently store as well as access large amounts of information.

- Since databases often contain confidential information, they are frequently the target of attacks.
- Thus, most web-based database interaction is carried on the server-side and is carefully controlled.
- The goal of the attacker is to breach this controlled interaction and get direct access to the database.
Web Security

Diagram showing a legitimate client, a web server, a database, and an attacker. The legitimate client communicates with the web server, which has protected, controlled access to the database. The attacker desires direct unprotected access to the database.
Web servers interact with most databases using the **Structured Query Language (SQL)**. Here are some operations it supports:

- **SELECT**: to express queries
- **INSERT, UPDATE, DELETE**
- **AND and OR operators**
- **UNION**: combines results of multiple queries into a single result.
Consider the table below called “news”, which stores news articles.

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>author</th>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Databases</td>
<td>John</td>
<td>(Story 1)</td>
</tr>
<tr>
<td>2</td>
<td>Computers</td>
<td>Joe</td>
<td>(Story 2)</td>
</tr>
<tr>
<td>3</td>
<td>Security</td>
<td>Jane</td>
<td>(Story 3)</td>
</tr>
<tr>
<td>4</td>
<td>Technology</td>
<td>Julia</td>
<td>(Story 4)</td>
</tr>
</tbody>
</table>

The SQL query: `Select * FROM news WHERE id = 3` will output the entire record (row) associated with id = 3.

The SQL query: `Select body FROM news WHERE author = "Joe"` will output Computers.
In an **SQL Injection**, an attacker inserts his own SQL commands in a data stream that is passed by the web server to the database. This results in the attacker gaining access to the database.

**Classic Example:** an SQL injection is used to let an attacker login.

```php
<?php
$query = 'SELECT * FROM users WHERE email = '' . $_POST['email'] . ''' . ' AND pwdhash = '' . hash('sha256',$_POST['password']) . ''';
$out = mysql_query($query) or die('Query failed: ' . mysql_error());
if (mysql_num_rows($out) > 0) {
    $access = true;
    echo "<p>Login successful!</p>";
}
else {
    $access = false;
    echo "<p>Login failed.</p>";
}
?>
```
If input is not properly validated then the following can happen: an attacker enters

Email: " OR 1 = 1: - -
Password: (empty)

The resulting query is

Select * FROM users WHERE email = " " OR 1=1; - - " AND pwdhash="e3 ..."
If input is not properly validated then the following can happen: an attacker enters

Email: " OR 1 = 1:  --
Password: (empty)

The resulting query is

\[
\text{Select * FROM users WHERE email = " " OR 1=1; -- " AND pwdhash="e3 ..."}
\]

- In SQL, query statements end in semicolon and "--" denote a comment. Hence, the rest of the line is ignored.
- Thus, the web server is querying records with no e-mail or where 1 = 1.
- Since the latter is always true, the query returns the entire users table as a result!
- This means that the attacker can successfully login.
Here's another example:

```php
<?php
    // Create SQL query
    $query = 'SELECT * FROM news WHERE id = ' . $_GET['id'];
    // Execute SQL query
    $out = mysql_query($query) or die('Query failed: ' . mysql_error());
    // Display query results
    echo '<table border=1>
';
    // Generate header row
    echo '<tr>
         <th>id</th><th>title</th><th>author</th><th>body</th>
    </tr>';
    while ($row = mysql_fetch_array($out)) {
        // Generate row
        echo '  <tr>
';
        echo '    <td>' . $row['id'] . '</td>
';
        echo '    <td>' . $row['title'] . '</td>
';
        echo '    <td>' . $row['author'] . '</td>
';
        echo '    <td>' . $row['body'] . '</td>
        </tr>
';
    }
    echo '</table>
';
?>
```
Once again, if the server-side code does not check to see whether the GET variable id is getting a valid input, bad things can happen!

For example, the attacker can plug the following into the GET variable:

http://www.example.com/news.php?id=NULL UNION SELECT cardno, first, last, email FROM users

This translates to the following SQL query:

SELECT * FROM news WHERE id = NULL UNION SELECT cardno, first, last, email FROM users

That is, the web server is also being asked to obtain info from another table called users.
Here’s how the results might look like:

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>author</th>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111-3333-5555-7777</td>
<td>Alice</td>
<td>All</td>
<td><a href="mailto:alice@example.com">alice@example.com</a></td>
</tr>
<tr>
<td>2222-4444-6666-8888</td>
<td>Bob</td>
<td>Brown</td>
<td><a href="mailto:bob@example.com">bob@example.com</a></td>
</tr>
</tbody>
</table>
Here’s how the results might look like:

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>author</th>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111-3333-5555-7777</td>
<td>Alice</td>
<td>All</td>
<td><a href="mailto:alice@example.com">alice@example.com</a></td>
</tr>
<tr>
<td>2222-4444-6666-8888</td>
<td>Bob</td>
<td>Brown</td>
<td><a href="mailto:bob@example.com">bob@example.com</a></td>
</tr>
</tbody>
</table>

**NOTE:**
- In order for an attacker to be successful in our two examples, he had to know something about the structure of the databases and the code used to query the databases.
- Attackers use many tactics to gather information about a database!

DEMOS
Defenses against Server-side Attacks

Developers

- The principle of *input validation*. A vast majority of web security issues can be prevented if developers made sure that everything a user inputs is checked for malicious behavior.

- Many languages feature built-in functions that strip inputs of dangerous characters. Developers should use ’em!
Administrators

- The principle of *least privilege*. Make sure that web servers are operating with the most restrictive permissions as possible.
- For example, web server should only have read privileges to the web site’s root directory, write privileges only to files and directories that absolutely need to be written to, and execute privileges only if necessary.
- This will limit the damage that a compromised web server can make.
Administrators

- The principle of *least privilege*. Make sure that web servers are operating with the most restrictive permissions as possible.
- For example, web server should only have read privileges to the web site’s root directory, write privileges only to files and directories that absolutely need to be written to, and execute privileges only if necessary.
- This will limit the damage that a compromised web server can make.
- Apply security updates and patches ASAP.
- Enforce safe browsing practices among its users.