

Unit - 5

Introduction to Advanced Server-Side Issues

- **Authentication:** Authentication is the process of determining the identity of a user based on the user's credentials. The user's credentials are usually in the form of user ID and password, which is checked against any credentials' store such as database. If the credentials provided by the user are valid, then the user is considered an authenticated user.
- **Authorization:** After successful authentication, deciding which resources a user can access based on their identity and checking whether the authenticated user has sufficient rights to access the requested resource is authorization.
- **Impersonation:** Impersonation is a process in which user accesses the resources(Ex:Files,DB) by using the identity of another user.
- There are four different kinds of Windows authentication options available that can be configured in IIS:
 - **Anonymous Authentication:** IIS runs all the users' requests using the identity of the IUSR_machine name account which is created by IIS. This is an example of Impersonation wherein user accesses the resources by using the identity of the another user. IIS doesn't perform any authentication check. IIS allows any user to access the ASP .NET application.
 - **Basic Authentication:** For this kind of authentication, a Windows user name and password have to be provided to connect. However, this information is sent over the network in plain text and hence this is an insecure kind of authentication. Basic Authentication is mostly supported by older, non-IE browsers.
 - **Digest Authentication:** It is same as Basic Authentication but for the fact that the password is hashed before it is sent across the network. However, to be using Digest Authentication, we must use IE 5.0 or above.
 - **Integrated Windows Authentication:** In this kind of authentication technique, passwords are not sent across the network. The application here uses either the kerberos or challenge/response protocols to authenticate users. Kerberos, a network authentication protocol, is designed to provide strong authentication for client-server applications. It provides the tools of authentication and strong cryptography over the network to help to secure information in systems across entire enterprise.

Authentication by IP Address in PHP

- Authentication by IP address involves allowing or denying access to a web application based on the user's IP address. While this method is not very secure and is not recommended for sensitive applications, it can be useful for basic access control in certain scenarios, such as restricting access to internal applications or resources.

```
<?php
function getUserIP() {
    if (!empty($_SERVER['HTTP_CLIENT_IP'])) {
        $ip = $_SERVER['HTTP_CLIENT_IP'];
    } elseif (!empty($_SERVER['HTTP_X_FORWARDED_FOR'])) {
        $ip = $_SERVER['HTTP_X_FORWARDED_FOR'];
    } else {
        $ip = $_SERVER['REMOTE_ADDR'];
    }
    return $ip;
}

$allowed_ips = array(
    '192.168.1.1',
    '203.0.113.5',
    '198.51.100.7'
);

$user_ip = getUserIP();

if (in_array($user_ip, $allowed_ips)) {
    echo "Access granted.";
    // Proceed with the rest of the application
} else {
    echo "Access denied.";
    // Optionally, redirect or log the attempt
    // header("Location: /access-denied.html");
    // exit();
}
?>
```



Basic File Handling Functions

1. Opening a File

```
<?php
$file = fopen("filename.txt", "mode");
?>
```

- Modes include:

- 'r' : Read-only. Starts at the beginning of the file.
- 'r+' : Read/Write. Starts at the beginning of the file.
- 'w' : Write-only. Opens and truncates the file to zero length or creates a new file for writing.
- 'w+' : Read/Write. Opens and truncates the file to zero length or creates a new file for reading/writing.
- 'a' : Write-only. Opens and writes to the end of the file or creates a new file for writing.
- 'a+' : Read/Write. Preserves file content by writing to the end of the file.

2. Reading from a File

- fread(): Reads a specified number of bytes from a file.

```
<?php
$content = fread($file, filesize("filename.txt"));
?>
```

- fgets(): Reads a single line from a file.

```
<?php
$line = fgets($file);
?>
```

- fgetc(): Reads a single character from a file.

```
<?php
$char = fgetc($file);
?>
```

3. Writing to a File

- fwrite(): Writes data to a file.

```
<?php
fwrite($file, "Hello, World!");
?>
```

- file_put_contents(): Writes data to a file, overwriting it or creating it if it doesn't exist.

```
<?php
file_put_contents("filename.txt", "Hello, World!");
?>
```

4. Closing a File

```
<?php
fclose($file);
?>
```

Additional File Handling Functions

1. Checking if a File Exists

```
<?php
if (file_exists("filename.txt")) {
    echo "The file exists.";
} else {
    echo "The file does not exist.";
}
?>
```

2. Deleting a File

```
<?php
unlink("filename.txt");
?>
```

3. File Uploads

To handle file uploads, PHP uses the `$ _FILES` array. A simple file upload script:

```
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $target_dir = "uploads/";
    $target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
        echo "The file ". basename($_FILES["fileToUpload"]["name"]). " has been upload
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
```

```
<?php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $target_dir = "uploads/";
    $target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
        echo "The file ". basename($_FILES["fileToUpload"]["name"]). " has been
uploaded.";
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
?>
```

4. Getting File Information

- o `filesize()`: Returns the size of a file.

```
<?php
$size = filesize("filename.txt");
?>
```
- o `filetype()`: Returns the type of a file.

```
<?php
$type = filetype("filename.txt");
?>
```
- o `pathinfo()`: Returns information about a file path.

```
<?php
$info = pathinfo("filename.txt");
?>
```

5. Reading an Entire File

- o `file_get_contents()`: Reads the entire file into a string.

```
<?php
$content = file_get_contents("filename.txt");
?>
```
- o `file()`: Reads the entire file into an array, with each element corresponding to a line in the file.

```
<?php
$lines = file("filename.txt");
?>
```

Error Handling

It's essential to handle errors when working with files to ensure that your application can gracefully manage issues such as missing files or permission problems.

```
<?php
$file = @fopen("nonexistentfile.txt", "r");
if (!$file) {
    echo "Failed to open file!";
}
?>
```

Example: Reading and Writing to a File

```
// Writing to a file
$file = fopen("example.txt", "w");
fwrite($file, "This is a sample text.");
fclose($file);

// Reading from a file
$file = fopen("example.txt", "r");
$content = fread($file, filesize("example.txt"));
fclose($file);

echo $content;
```

Integrated Windows Authentication

Integrated Windows Authentication (IWA), also known as Windows Integrated Authentication, is a secure form of authentication in which the user's Windows credentials are used to authenticate the user. It is typically used in intranet environments where both the server and the client are within the same Windows domain.

Implementing Integrated Windows Authentication in PHP requires a bit of setup, as PHP does not natively support IWA. However, you can configure your web server (e.g., Apache or IIS) to handle the authentication part and then pass the authenticated user information to your PHP application.

Implementing IWA in PHP

Step 1: Configure the Web Server

For IIS (Internet Information Services)

1. **Enable Windows Authentication:**
 - Open the IIS Manager.
 - Navigate to your site or application.
 - In the "Features View," double-click on "Authentication."
 - Enable "Windows Authentication" and disable other authentication methods like "Anonymous Authentication."
2. **Pass User Information to PHP:**
 - Ensure that the authenticated user information is available to your PHP application via the `$_SERVER` superglobal, typically in `$_SERVER['REMOTE_USER']`.

For Apache

1. **Install and Enable mod_authnz_ldap (if using LDAP):**
 - Ensure the `mod_authnz_ldap` and `mod_auth_basic` modules are installed and enabled. You might need to add these to your `httpd.conf` or `apache2.conf` file:

```
LoadModule authnz_ldap_module modules/mod_authnz_ldap.so
LoadModule auth_basic_module modules/mod_auth_basic.so
LoadModule authz_user_module modules/mod_authz_user.so
```

2. **Configure the Apache Virtual Host:**
 - Edit your Apache configuration file (e.g., `httpd.conf` or a specific site configuration file).

```
<Directory "/path/to/your/application">
    AuthType SSPI
    AuthName "SSPI Authentication"
    SSPIAuth On
    SSPIAuthoritative On
</Directory>
```

```
SSPIOfferBasic On
Require valid-user
</Directory>
```

- Make sure you have the `mod_auth_sspi` module installed for Apache on Windows.

3. Restart Apache:

- Restart your Apache server to apply the changes.

Step 2: Access the Authenticated User in PHP

Once the server is set up to handle IWA, it will pass the authenticated user information to PHP. You can access this information using the `$_SERVER` superglobal.

```
<?php
session_start();

if (isset($_SERVER['REMOTE_USER'])) {
    $username = $_SERVER['REMOTE_USER'];
    echo "Authenticated user: " . htmlspecialchars($username);

    // You can store the username in a session variable if needed
    $_SESSION['username'] = $username;
} else {
    echo "No authenticated user.";
}
?>
```