

# Secure Authentication and Authorization

Introduction .....	1
Planning for Authentication and Authorization .....	2
Data Classification Categories .....	2
Data Inventory .....	2
User Classification Group.....	3
Application Functionality Inventory .....	3
Involving all Project Stakeholders.....	3
Implementing Authentication and Authorization .....	3
Establish and Maintain Unique Application Session ID .....	4
Identification of Unique (Registered) Users / Login .....	5
Enforce Data and Application Functionality Access Control.....	8
Terminating Authenticated Sessions Through Logout and Time Based Expiration .....	10
Using the AuthenAutho Class.....	13
Conclusion .....	13

## Introduction

Building a web application in 2008, typically will require implementing some sort of social networking functionality that allows users to interact with you or each other. In order to benefit from the social networking functionality of the application, visitors must to identify themselves through their registered username and password for their content to be attributed to them. As designers, developers, and auditors of these social web applications, we are responsible for ensuring that the data collected from the users, including their registered identity, remain under their control and adhere to the appropriate corporate privacy policies.

This document is intended for designers and developers of web application. It describes both the business and technical steps required to build a secure authentication and authorization system. Some of the content includes PHP code samples which require a good understanding of the programming language and web application architecture.

## Planning for Authentication and Authorization

Conceptually, a secure authentication and authorization model will provide a centralized mechanism for developers to implement the following concepts:

- Definition of security group and/or user access rights to application functionality and data
- Ability to establish and maintain unique application session
- Identification of unique (registered) users
- Mapping of session ID to user ID
- Mapping of users to their assigned security group
- Enforce defined application functionality access restrictions
- Enforce defined data access restrictions
- Ability to terminate an authenticated session through Logout
- Enforce time-based automatic termination of authenticated sessions

### ***Data Classification Categories***

First and foremost, it is important to understand the type of data being protected. Project stakeholders must compile a list of data classification groupings tailored to the type of data being used and stored by the application. As an example, the following data groups might exist for a "typical" social network, however, additional groups might be required depending on your application type.

**Restricted** - assigned to data which only the data owner should have access to read or modify (eg: credit card number, password).

**Confidential** - assigned to data where the user can choose who they want to share the data with (eg: phone number, home address).

**Public** - assigned to data that can be viewed by anyone (eg: username, photo, online status).

### ***Data Inventory***

A data inventory must now be created to establish an all inclusive list of data fields and types available to the application through the data store (e.g., database, structured flat files, etc). Once the data inventory has been established, review each data item and assign it to a data classification group. It is important to enlist a data privacy expert or legal representative to help with data classification; the work done here will determine additional requirements for technical safeguards to protect each data classification group.

**Please note** that most social networks will allow the user to define the data classification assigned to the data they store in the application. If this is a requirement for your application, additional work will have to be done to establish

default classification levels and define which data items will receive user defined classification.

## ***User Classification Group***

The next step is to define user classification groups. These groups will help establish access rights to application functionality and data. The following groups should be defined for any web application. Additional groups, such as author, moderator, or contributor can be added based on the requirements for the web application.

**Unauthenticated user** - defines any visitor that has not yet provided their username and password.

**Authenticated user** - defines any registered user who has provided their username and password.

**Administrator** - defines any registered user who has been granted administrative access.

## ***Application Functionality Inventory***

Next, create an application functionality inventory of all of the functionality available in the web application. Once the inventory is complete, assign user classification groups to each individual piece of application functionality. The following is a short list of examples which might be included in the list and their assigned user classification groups.

**Search** - Authenticated user

**Modify Profile** - Authenticated user

**View blog** - All visitors

**User Administration** – Administrator

## ***Involving all Project Stakeholders***

Each web application will vary in its data classification and user classification requirements based on the overall business and technical requirements. As such, it is imperative to involve all of the project stakeholders in the definition of business rules for data classification, functionality authorization, and data privacy enforcement.

## **Implementing Authentication and Authorization**

With a clear understanding of the authentication and authorization needs of the web application, it is time to start defining the technical components required for

protecting the data and application functionality. This paper uses code examples to illustrate key components of the authentication and authorization model. The code is written in PHP with a MySQL database. Any assumptions on application functionality outside of the scope of the authentication and authorization models are indicated in the code comments.

## ***Establish and Maintain Unique Application Session ID***

Because HTTP is a stateless protocol, web application developers need to leverage session identifiers stored in client cookies to map unique requests to their originator. Most popular web application programming languages have some sort of built-in session identification mechanism:

**PHP** - PHPSESSID cookie value managed through the PHP session object.

**Java** - The JSESSIONID cookie value managed through the Java HttpSession object.

Most web application programming languages offer the ability to store additional information in the cookie/session object. While this might be appealing for simplicity's sake, this practice is discouraged, as the cookie object can be modified by a malicious attacker in an attempt to gain access to data and application functionality belonging to another user.

The following snippet of code creates the AuthenAutho class and assigns all default values. These default values must initialize the session as unauthenticated and reflect the lowest authorization level available in the model.

```
class AuthenAutho {
    private $sessionId;
    private $sessionAuthoLevel;
    private $user_id;
    private $AppPermissions;
    private $dataPermissions;
    public function AuthenAutho() {
        /* Provides the class startup functionality for the
        Object.
        */
        # Setting up some VERY basic Application Permissions.
        Your needs and implementation will certainly vary.
        $this->AppPermissions = array(
            "search"=>"authenticated",
            "modify_profile"=>"authenticated",
            "view_blog"=>"all",
            "user_admin"=>"administrator"
        );
    }
}
```

```

# Setting up some VERY basic Data Permissions. Your
needs and implementation will certainly vary.
$this->dataPermissions = array(
"modify_profile"=>"confidential",
"detail_profile"=>"restricted",
"short_profile"=>"public"
);
# Set the default authorization level. Your needs and
implementation will certainly vary.
$this->sessionAuthoLevel = "unauthenticated"; # Set
the default user_id.
$this->user_id = 0;
# Connect to database
mysql_connect("localhost","username","password");
# Select the project_db as the default database
mysql_select_db("project_db");
# Start the PHP session object.
session_start();
#Assign the sessionid value to class object $sessionId
$this->sessionId = session_id();
return;
}

```

## ***Identification of Unique (Registered) Users / Login***

The general purpose of the login function is to identify a unique user through username and password validation; then associate an application session ID with an application user ID. The login function should then establish any session specific data items such as user id, default authorization level, etc. Additionally, there are several considerations to ensure this functionality is secure.

**Fail to unauthenticated** in the event of any failure in the code or credential validation, the application will default to an unauthenticated session.

**Validate Input** - use built-in functions to ensure the SQL statement is not subject to SQL Injection vulnerabilities.

The following snippet of code creates a login function which will accept the username and password as input and return TRUE for a successful login or FALSE in the event of a failure.

```

public function login($username,$password) {
/* Provides validation of provided user credentials
against the user_table table in the database.

```

```

* $username: username from the login form submitted
over HTTPS.
* $password: password from the login form submitted
over HTTPS.
* returns TRUE for successful login, FALSE for failed
login.
*/
$sql = "SELECT user_id,user_level FROM user_table
WHERE username = \"\" . addslashes($username) . "\"" AND
password=\"\" . addslashes($password) . "\"" LIMIT 1";
/* SQL Statement explanation:
* The SQL statement queries the user_table for a
user_id where the provided username and password
matches the values stored
* Note: the addslashes() function will escape any
MySQL special characters such as ', ", and %.
*/
# Use the mysql_num_rows PHP object to get the number
of rows returned by # the query.
$numResults = mysql_num_rows($sql);
# If there is 1 row returned in the query, then we
successfully found a # user_id with username and
password matching those we provided.
if($numResults==1) {
    # Fetch the results from the SQL query into
object $result.
    $result = mysql_fetch_object($sql);
    # Assign the value of the user_id database field
to $user_id.
    $this->user_id = $result->user_id;
    if($this->createAuthenticatedSession($this-
>user_id)==TRUE) {
        if($result->user_level=="administrator") {
            $this->sessionAuthoLevel =
"administrator";
        } else {
            $this->sessionAuthoLevel =
"authenticated";
        }
        # if the authenticated session was
successfully written.
        return TRUE;
    } else {
        # if there was a error along the way, we
want to fail login.
        return FALSE;
    }
}

```

```

} else {
    #Any other condition we want to fail login.
    return FALSE;
}
#Fail login just in case something went wrong.
return FALSE;
}
private function createAuthenticatedSession($user_id)
{
    /* Issues a new session Id and writes an entry in the
    authenticatedSessions database table
    * $user_id: valid user id from the user_table database
    table with which this session will be associated
    */
    $sql = "DELETE FROM authenticatedSessions WHERE
    sessionId = \"\" . addslashes($this->sessionId) . \"\"
    LIMIT 1";
    # Delete any entries in the authenticatedSessions
    database table matching # the current sessionId, just
    in case.
    mysql_query($sql);
    # Regenerate the session id. This will destroy any
    custom entries you # have for the $_SESSION object.
    session_regenerate_id();
    # Assign the new session id back to the class object
    $sessionId
    $this->sessionId = session_id();
    $sql = "INSERT INTO authenticatedSessions
    (sessionId,user_id,established,last_renew) VALUES (\",\" .
    addslashes($this->sessionId) . "\",\" .
    addslashes($user_id) . "\",\" . time() . \",\" . time() .
    \")";
    # Insert the sessionId/user_id combination into the
    authenticatedSessions # table
    if(mysql_query($sql)==TRUE) {
        return TRUE;
    } else {
        return FALSE;
    }
}
return FALSE;
}

```

The following snippet of code returns TRUE if this session has successfully executed the login() function by checking the authenticatedSessions database table.

```

public function isLoggedIn() {

```

```

/* Provides authentication status for the current
session id.
* returns TRUE is the session is in the
authenticatedSessions table.
*/
$sql = "SELECT * FROM authenticatedSessions WHERE
sessionId = \"\" . addslashes($this->sessionId) . "\""
LIMIT 1";
$query = mysql_query($sql);
$numResults = mysql_num_rows($query);
if($numResults==1) {
    $results = mysql_fetch_object($query);
    if($this->checkSessionTime($results)==TRUE) {
        return TRUE;
    } else {
        return FALSE;
    }
} else {
    return FALSE;
}
return FALSE;
}

```

**Please note** that these code samples are based on the VERY rudimentary data classification and application functionality authorization maps. Some additional levels of complexity may be introduced by a more complicated application with granular data field access control requirements. As such, it is important for all technical and non-technical project stakeholders to have a clear understanding of the specific requirements for data privacy and application functionality access control.

### ***Enforce Data and Application Functionality Access Control***

Building a global way of validating access control to data and application functionality will allow for a centralized point from which to manage and implement the business rules for data classification and the defined application functionality mapping. A well designed access control mechanism will implement the following concepts.

- Fail unauthorized** in the event of any failure in the code or authorization verification, the application will default to an unauthorized state.
- Authorize access** based on a successful match of the user security level (eg: authenticated) or user identity against the permissions assigned to the requested object.

The following code snippet implements the authorization rules which were defined as examples for the data classification levels and the application

authorization groups during the security model definition and requirements phase.

```
public function
isAuthorized($requestType,$requestObject,$owner_user_id=0) {
    /* Provides an authorization check for the specific
    requestType and * requestObj against the current
    session permissions.
    * $requestType: either application or data.
    * $requestObject: the application functionality or
    data group being.
    * $owner_user_id (optional): user id of the data which
    is being accessed.
    * returns TRUE if authorized, FALSE otherwise
    */
    switch($requestType) {
        case "application":
            if(isset($this->AppPermissions[$requestObject]))
            {
                if($this->AppPermissions[$requestObject]==$this->sessionAuthoLevel) {
                    return TRUE;
                } else {
                    return FALSE;
                }
            } else {
                return FALSE;
            }
            break;
        case "data":
            if(isset($this->dataPermissions[$requestObject])) {
                switch($this->dataPermissions[$requestObject]) {
                    case "confidential":
                        if($owner_user_id!=0&&$owner_user_id==$this->user_id) {
                            return TRUE;
                        } else {
                            return FALSE;
                        }
                    break;
                    case "restricted":
                        if($owner_user_id!=0) {
                            /*
```

```

        * The way restricted information
        is given permissions is beyond the
        scope of this paper.
        * A function should be called that
        will return TRUE or FALSE given
        the $requestObject,
        $owner_user_id, and $this->
        user_id.
        */
        return TRUE;
    } else {
        return FALSE;
    }
    break;
    case "public":
    return TRUE;
    break;
    default:
    return FALSE;
    break;
    }
    } else {
        return FALSE;
    }
    break;
    default:
    return FALSE;
    break;
}
return FALSE;
}

```

**Please note** that the code samples are based on a VERY rudimentary data classification and application functionality authorization map. Some additional difficulties may be introduced by a more complex application with granular data field access control requirements. As such, it is important for all technical and non-technical project stakeholders to have a clear understanding of the specific requirements for data privacy and application functionality access control.

### ***Terminating Authenticated Sessions Through Logout and Time Based Expiration***

Allowing for an authenticated session to be terminated by the user or to expire after a set period of inactivity minimizes the window of time in which identity based (cookie based) attacks will be effective. Additionally, changing the session key for every authenticated user every 5 minutes will further reduce the time during which an attacker can launch an identity based attack.

The following code snippet shows the logout function which simply terminates the session and deletes the entry from the database. The checkSessionTime function is a code implementation of the business rules that require a session to be terminated after 30 minutes of inactivity and the session id to be changed every 5 minutes.

```
public function logout() {
    /* Terminates the authenticated session.
    * returns TRUE.
    */
    $sql = "DELETE FROM authenticatedSessions WHERE
    sessionId = \"\" . addslashes($this->sessionId) . \"\"
    LIMIT 1";
    # Delete any entries in the authenticatedSessions
    database table matching # the current sessionId, just
    in case.
    mysql_query($sql);
    # Regenerate the session id. This will destroy any
    custom entries you # have for the $_SESSION object.
    session_regenerate_id();
    # Assign the new session id back to the class object
    $sessionId
    $this->sessionId = session_id();
    $this->sessionAuthoLevel = "unauthenticated";
    $this->user_id = 0;
    return TRUE;
}

private function checkSessionTime($sessionObj) {
    /* Checks the established time to cycle session Id's
    every 5 minutes and * checks the last_used time to
    enforce 30 minute timeout.
    * $sessionObj is the object returned by
    mysql_fetch_object.
    */
    $expireTime = mktime(date("H",$sessionObj->
    established),date("i",$sessionObj->
    established)+30,date("s",$sessionObj->
    established),date("m",$sessionObj->
    established),date("d",$sessionObj->
    established),date("Y",$sessionObj->established));
    $renewTime = mktime(date("H",$sessionObj->
    last_renew),date("i",$sessionObj->
    last_renew)+5,date("s",$sessionObj->
    last_renew),date("m",$sessionObj->
    last_renew),date("d",$sessionObj->
    last_renew),date("Y",$sessionObj->last_renew));
    if($expireTime<time()) {
```

```

$sql = "DELETE FROM authenticatedSessions WHERE
sessionId = \"\" . addslashes($this->sessionId) .
\"\" LIMIT 1";
# Delete any entries in the authenticatedSessions
database table matching the current sessionId,
just in case.
mysql_query($sql);
# Regenerate the session id. This will destroy
any custom entries you have for the $_SESSION
object.
session_regenerate_id();
# Assign the new session id back to the class
object $sessionId
$this->sessionId = session_id();
$this->sessionAuthoLevel = "unauthenticated";
$this->user_id = 0;
return FALSE;
} else {
    $sql = "UPDATE authenticatedSessions SET
    established = " . time() . " WHERE sessionId =
    \"\" . $this->sessionId . \"\" LIMIT 1";
    if(mysql_query($sql)==FALSE) {
        return FALSE;
    }
}
if($renewTime<time()) {
    session_regenerate_id();
    $sql = "UPDATE authenticatedSessions SET
    sessionId = \"\" . addslashes(session_id()) . "\",
    last_renew = " . addslashes(time()) . " WHERE
    sessionId = \"\" . addslashes($this->sessionId) .
    \"\" LIMIT 1";
    $this->sessionId = session_id();
    if(mysql_query($sql)==TRUE) {
        return TRUE;
    } else {
        return FALSE;
    }
} else {
    return TRUE;
}
return FALSE;
}

```

## Using the AuthenAutho Class

With a solid authentication and authorization model in place, we will now explore how each function should be used throughout the project code.

```
# Create an instance of the AuthenAutho class.
$AuthenAutho = new AuthenAutho();
# Check to see if the session is authenticated
if($AuthenAutho->isLoggedIn()==TRUE) {
    # authenticated Session
} else {
    # Unauthenticated Session
}
# Check login credentials
if($AuthenAutho-
>login("myusername","mypassword")==TRUE) {
    # Successful authentication
} else {
    # Failed authentication
}
# Check to see if the user is authorized to use
functionality or get data
# An imaginary owner ID of 1 is assigned to $myUUID
$myUUID = 1;
if($AuthenAutho-
>isAuthorized("application","modify_profile",$myUUID==T
RUE) {
    # Is authorized
} else {
    # Is not authorized
}
# Terminate authenticated session (logout)
$AuthenAuth->logout();
```

**Please note** that this implementation of the authentication and authorization model does not require the developers to worry about time based session id expiration. The AuthenAutho class contains two private functions which are called throughout the other public functions to ensure the session expiration and session refresh rules are applied.

## Conclusion

Solid Authentication and Authorization are the cornerstones upon which secure applications are built. From the planning stage where all project stakeholders have a voice in deciding upon the best way to implement a model to the code level implementation of the model, each step is critical in establishing and

maintaining the trust of web users. There are no easy solutions, such as Web Application Firewalls (WAF), input validation, output encoding, or automated code analysis to protect data and functionality against a poor authentication and authorization model or an error in implementation. As such, the purpose and technical details of the design must be communicated to all project stakeholders.