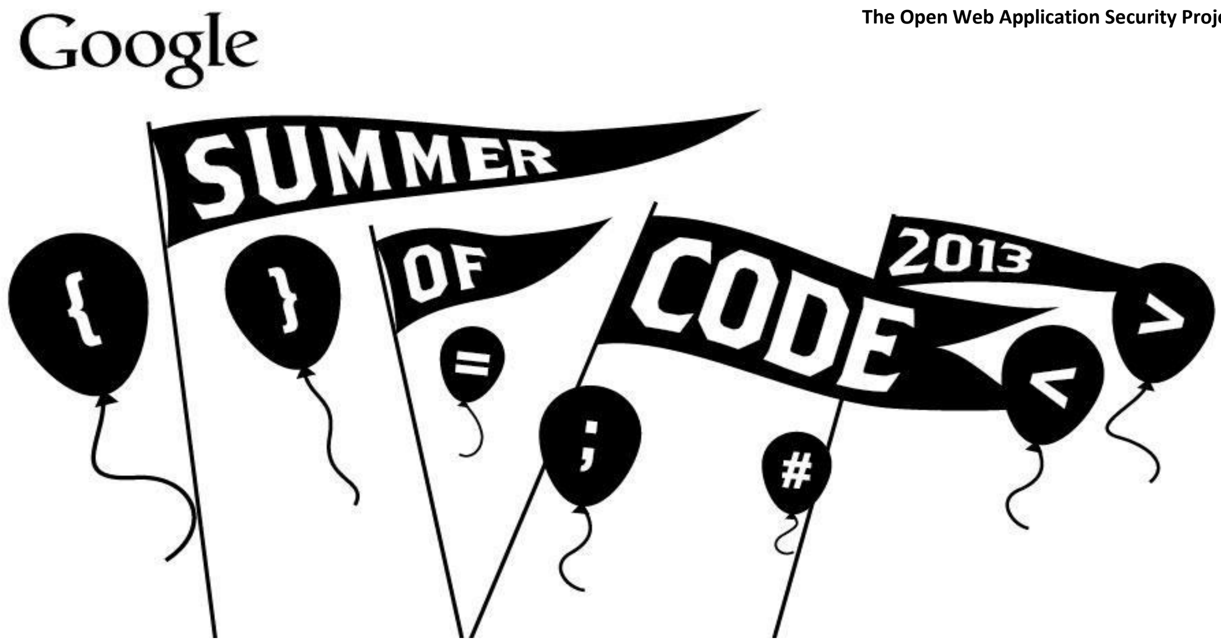
OWASP

The Open Web Application Security Project



OWASP PHP Security Project

GSOC 13 Proposal

Student
Name: Rahul Chaudhary
E-Mail: rac130@pitt.edu

Mentor
Name: Abbas Naderi
E-Mail: abiusx@owasp.org

Table of Contents

Executive Summary........2

Introduction........2

Libraries vs. Framework........2

Proposed Framework........2

Stand-Alone Libraries........3

- PHP Secure Database Library........3
- PHP Session Management Library........3
- PHP Password Management Library........4
- PHP Simple Authentication Library........5
- PHP Extended Authentication Library........5
- Secure PHP Time and Randomness Management Library........6
- Secure PHP HTTP Request Handling Library........7
- Secure PHP Static HTTP Response Handling Library (aka File Downloader) 7
- PHP-RBAC Library........8
- Secure Application Configuration and State........9
- PHP Error Handler Library........10
- Extra Libraries........10

References........11

Timeline........11

About Me........12

3rd Party Involvement........13

Contact Information........13

Executive Summary

OWASP PHP Security project's objective is to secure PHP libraries, and provide a full featured framework of standalone libraries for secure web applications in PHP, releasing them both as separate decoupled libraries and as a whole secure web application framework; where sample configuration and usage can be observed. Many aspects of this project are already handled, and are either added or being added to OWASP.

Introduction

PHP is extremely popular in web applications and thus OWASP has proposed to create libraries and frameworks to enhance security in these applications. Most imminent threats described in OWASP TOP 10 and few more are the first one's that are my objective in this project to resolve and implement. This library will then be expanded as new requirements and threats arises.

Libraries vs. Framework

There are a lot of PHP frameworks out there, and a few new ones pop out every day. People get used to them, and are reluctant to change. Many of these frameworks are matured over the time, and do not have core security features, but most of the security is pasted to the framework in the course of time. If we proposed a new framework - even though OWASP branded - only a few new developers (mostly security focuses) would turn to use it. If we developed a lot of libraries, developers would not trust them a lot, and wouldn't like to integrate all of them into their code. Frameworks would take more time to adopt them, because of the hardships in configuration and usage.

If we propose both, we attract three sets of developers:

- Fresh users, looking for a good framework to start building their applications.
- Enterprises, which employ their custom frameworks for creating applications.
- Other Frameworks, which will slowly but easily integrate our libraries with their code.

Proposed Framework

A proposed framework demonstrating how to properly mix all those libraries with other common settings and features of web application frameworks will be developed. The framework is not just a model, it can be used for real-life applications and will be heavily extensible.

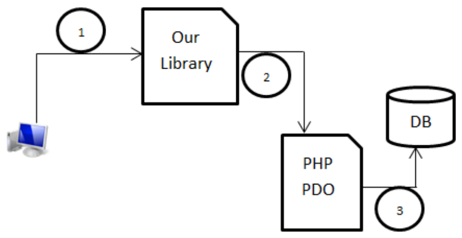
OWASP will take charge of incubating and expanding the framework, both in impact and in features; so that in the long run it takes over other common insecure frameworks currently in use.

Stand-Alone Libraries

1. PHP Secure Database Library

This library will be compatible with PHP PDO, but disallow insecure operations (such as concatenation of values in any form) with it. We are going to enforce prepared statements for all data that is to be sent to the database engine, and enforce whitelisting (via taint tracking) of all SQL parameters (such as limit and order by) where prepared data are not supported by the back-end engine.

A base library will provide all these features abstracted from database engines, derived libraries will be created for each common database engines and new adapters for SQL databases will be added.



1. User sends a SQL statement.
2. Request is processed by our library where we format the statement such as using prepared statement.
3. PHP-PDO will then further format this data and will query the DB. The result will be returned back to our library and then back to user.

2. PHP Session Management Library

This library will take over session handling from PHP. PHP's internal session handling has many flaws, and most of its usages are insecure because due to backward compatibility, lots of new functionality are added as extra parameters to certain functions, and are never used by the developers (e.g session_start).

Best practices of secure session management will be handled by the library, and session data will be handled by a database (or file via sqlite). Handling of HTTP Cookies and rolling of sessions, as well as high entropy random values are intended for this library.

The outcome of this library would be much better performance for session handling, best practices of session handling easily integratable with user management libraries and awareness of other applications on the level of elevation and session.

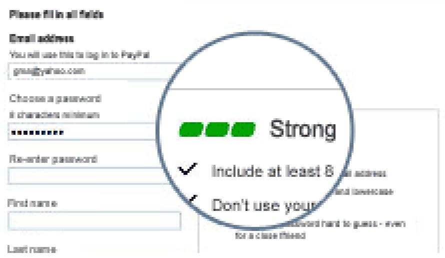
This library implementation would include these:

- Use of cryptographically secure pseudo-random number generators that are hard to guess.
- Allowing single-login per user or multiple-login per user by flexible configuration, as well as tracking of visitor activities.
- Enforcing session timeout for active sessions and inactive session
- Rolling of sessions on elevation change

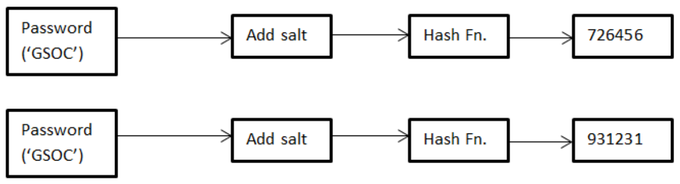
3. PHP Password Management Library

This would be a lightweight library for:

i. **Enforcing secure passwords:** There are many sources that cite most common passwords that must be avoided. e.g. here and here. Also many reliable sources including OWASP provides general guidelines for password length and complexity. This library would implement these guidelines for forcing users to keep a strong and secure password by combining heuristics, entropy, pattern recognition etc.



ii. **Password Storage and Salting:** Password theft on many websites discloses the password storage scheme, the weakness of that scheme, and often discloses a large population of compromised credentials that can affect multiple web sites or other applications. OWASP provides a cheat sheet and general guidelines here for developers to build a secure storage scheme. This library would follow these guidelines and will provide the developers with a ready implementation of these guidelines.

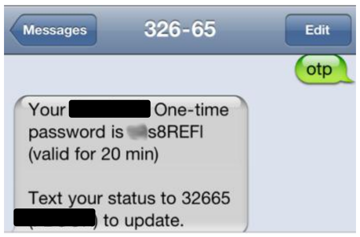


As we can see that even by using same passwords, we achieve different hashes.

iii. **Password generation scheme updating:** The guidelines for secure passwords keeps updating as new problems are introduced. Thus our library's implementation would have functions that makes this task easy for developers to change the scheme which generates good cryptographically secure passwords. e.g. Recent Linkedin attack.

iv. **Password Reusability:** This library would restrict users from using old passwords as new passwords. This could be implemented by keeping the hash of user's last 'n' passwords and checking if the new password matches any of those hashes.

v. **Transaction Passwords:** A second way of authentication (such as a separate transaction password or a temporary password sent to mobile phones) is sometimes needed for secure and sensitive transactions. e.g. online money transfer.



otp

Your [redacted] One-time password is [redacted]s8REFI (valid for 20 min)

Text your status to 32665 [redacted]) to update.

4. PHP Simple Authentication Library

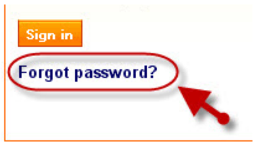
Basically, this would be a simple user management and authentication library. This library employs the session management library if present, and uses native PHP sessions otherwise. Password management library (at least two of its components) are required by this one. Other set of important distinctions:

i. **Simplified user management:** Users in this library are basically IDs and usernames, with a few fields for password management. Policies on usernames can be set, and if no more user management is required, user information (i.e profile) could be linked to this data.

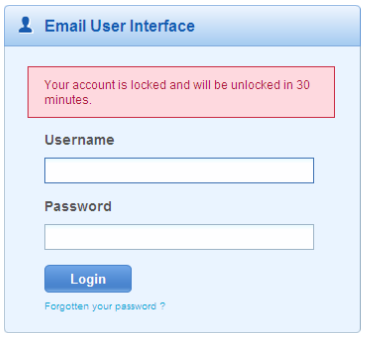
5. PHP Extended Authentication Library

This library will be created on top of the simple authentication library, and provide a bunch of new features:

i. **Forgot Password Feature:** Allowing users to reset their password using emails and reset tokens, and temporary passwords.



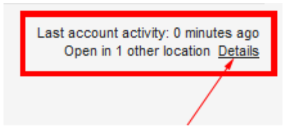
ii. **Brute-Force Locks:** Account will be locked if maximum number of attempts are over, and the duration of locks are adequately increased.



Your account is locked and will be unlocked in 30 minutes.

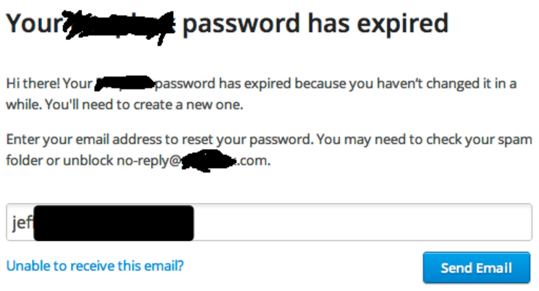
iii. **Temporary Passwords:** Used when creating and activating accounts by a third party.

iv. **Login Tracks:** Heuristics to determine login location, and take precautions when login is attempted from another location.


Last account activity: 0 minutes ago
Open in 1 other location Details

(**Fig:** Gmail's account activity)

v. **Password Expiry:** Sets of policies enforcing password change after a duration of time or events.


Your [redacted] password has expired

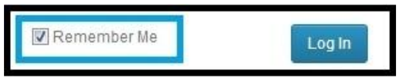
Hi there! Your [redacted] password has expired because you haven't changed it in a while. You'll need to create a new one.

Enter your email address to reset your password. You may need to check your spam folder or unblock no-reply@[redacted].com.

jef[redacted]

vi. **Account Activation and Confirmation:** When accounts newly created, the email should be verified.

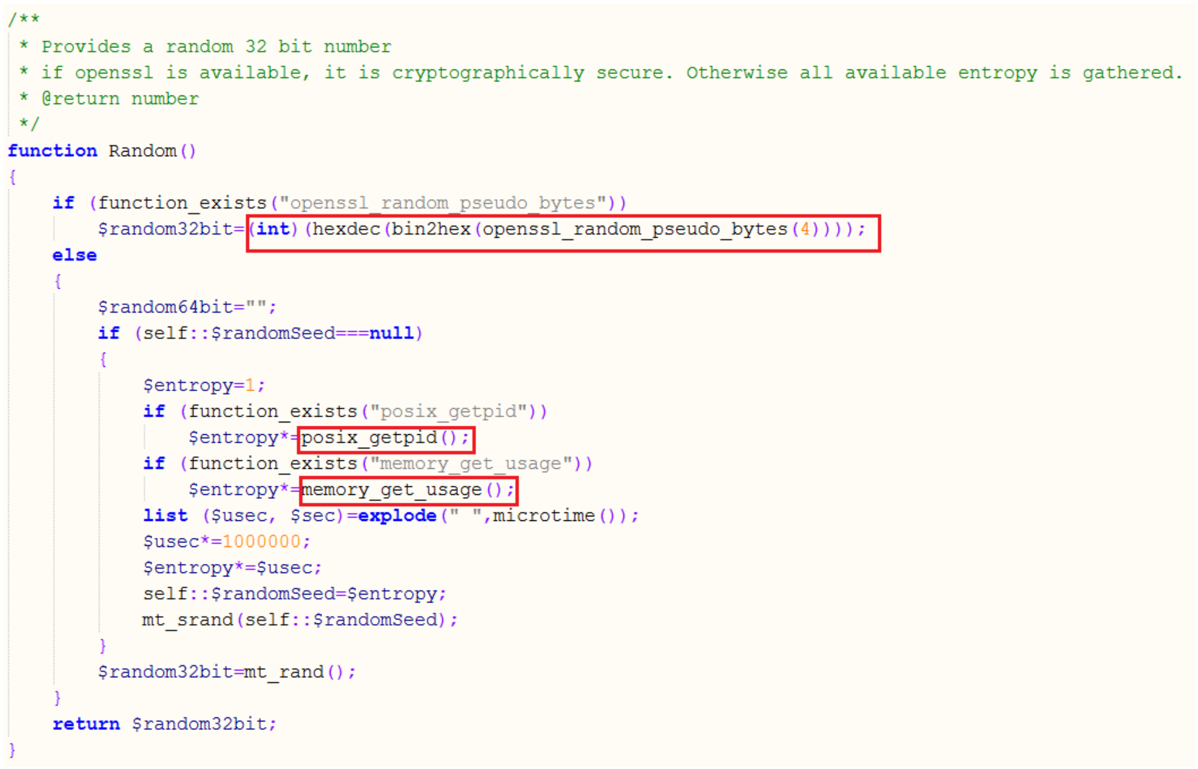
vii. **Remember Me:** Proper implementation of remember me feature.



Developers can use this library to easily manage all user-management and authentication aspects of their application. Valid and secure implementations of all these features are very rare in systems, and it has been a constant issue in OWASP Top Ten list.
This library simplifies the whole process, and provides sample UI models to be used with them. Proper UI is a very important part of authentication process, providing users with needed functionality and errors.

6. Secure PHP Time And Randomness Management Library

This library will provide functionality for date/time handling as well as secure randomness. It is a core library and should replace many usages of other functions in the code. Secure randomness would be generated using a mixture of features, based on hardware and operating system and PHP support.

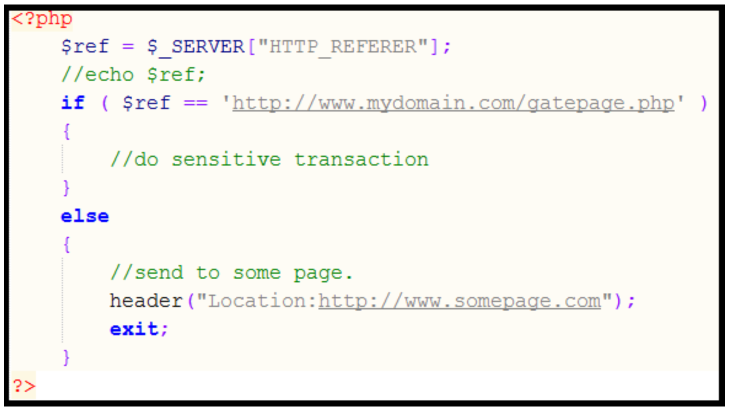


```
/**
 * Provides a random 32 bit number
 * if openssl is available, it is cryptographically secure. Otherwise all available entropy is gathered.
 * @return number
 */
function Random()
{
    if (function_exists("openssl_random_pseudo_bytes"))
        $random32bit=(int)(hexdec(bin2hex(openssl_random_pseudo_bytes(4))));
    else
    {
        $random64bit="";
        if (self::$randomSeed===null)
        {
            $entropy=1;
            if (function_exists("posix_getpid"))
                $entropy*=posix_getpid();
            if (function_exists("memory_get_usage"))
                $entropy*=memory_get_usage();
            list ($usec, $sec)=explode(" ",microtime());
            $usec*=1000000;
            $entropy*=$usec;
            self::$randomSeed=$entropy;
            mt_srand(self::$randomSeed);
        }
        $random32bit=mt_rand();
    }
    return $random32bit;
}
```

(**Fig:** This is the randomness function from jFramework library. Our library would be similar to this.)

7. Secure PHP HTTP Request Handling Library

HTTP Request is user input. Many developers forget this fact and tend to rely on it as a trustable source and configure many aspects of their applications based on values of $_SERVER (most of which are set using HTTP request). While not all values under $_SERVER are unreliable, some of the values such as 'QUERY_STRING', 'HTTP_REFERRER' etc are entirely arbitrary information sent by the client. This library provides wrappers which securely process these data and hand them to user, and replaces the $_SERVER values that are insecure with objects that throw exceptions when cast to string (e.g. in HTTP_HOST), so that developers can no longer directly access them.



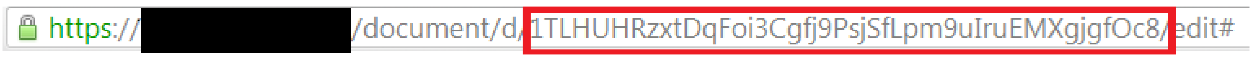
```php
<?php
    $ref = $_SERVER["HTTP_REFERER"];
    //echo $ref;
    if ( $ref == 'http://www.mydomain.com/gatepage.php' )
    {
        //do sensitive transaction
    }
    else
    {
        //send to some page.
        header("Location:http://www.somepage.com");
        exit;
    }
?>
```

As can be seen that this code entirely depends on the HTTP_REFERRER value to do a sensitive transaction. A potential attacker can easily spoof this variable and can trick the server to perform sensitive transaction.

8. Secure PHP Static HTTP Response Handling Library (aka File Downloader)

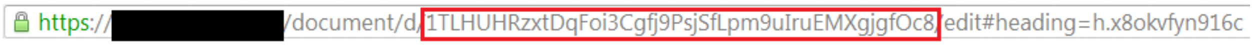
Currently web servers are largely dependent on "security by obscurity". They feed their clients with static files by generating a large random number for that file in hopes that until the hackers are unable to find the long random number, their files are safe. However OWASP has pointed out that for users who can save a particular location/URL of file can also access the file afterwards because they now know the random numbers and the file location is static. This operation should be handled by the application and not the web server, otherwise no access control check, bandwidth enforcement and other similar necessity could be performed at this point (a common flaw in many applications).

(**Fig1:** File location for 1st try)



https://[redacted]/document/d/1TLHUHRzxtDqFoi3Cgfj9PsjSfLpm9uIruEMXgjgfOc8/edit#

(**Fig2:** File location for 2nd try)



https://[redacted]/document/d/1TLHUHRzxtDqFoi3Cgfj9PsjSfLpm9uIruEMXgjgfOc8/edit#heading=h.x8okvfyn916c

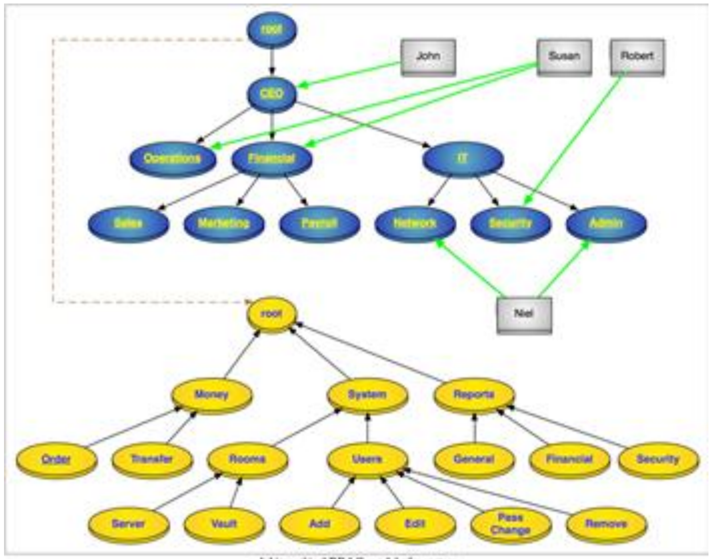
As we can see that in both the attempts, we see the same random number is used for this file which might be dangerous. Our library implementation will handle this. Also, functions such as “resume download” used by download managers are directly affected the way these functions are implemented. Proper implementation of this feature is of vital importance, otherwise download managers and old browsers will be unable to use the application properly.

9. PHP-RBAC Library

All web applications need some access control mechanism to control which entity accesses what resources. The need for PHP RBAC model rose when other models such as DAC/ACLs became too complicated to manage. Prior models like DAC/ACLs are resource based access control where for each resource the list of entities that can access it has to be defined. It can be easily seen here that with increase in number of resources, the complexity in management increases exponentially. Even if a single entity is deleted/added, existing/new access rules have to be added in each resource.

With RBAC, developers get a very natural hierarchy while developing their applications. They can create a central policy where they create specific roles and then they map each role with resources they think are necessary for them to have. This way the management task becomes less painful as now they only have to change the central policy to remove/add resources from a role. Adding/Deleting users also is very easy as they now have only to assign a particular user a role which suits his need.

Also the choice for RBAC over ACL is wise even if the organization is small and there is not much complexity in maintenance. The primary reason is the one discussed above. Also with many sub-roles inside a role, RBAC can be forced to behave as ACLs. RBAC can also behave in a manner where higher roles can also access resources in lower roles, thus masquerading as other access controls such as Bell LaPadula model and Biba Model.



A hierarchical RBAC model of a system
Blue: roles, Gray: users, Yellow: permissions

(**Fig:** http://phprbac.net/)

There are a number of reasons for RBAC not being implemented vigorously in all web-applications despite its benefits. Primary reason is because of the need of more fined grained controls. Often in large organizations people need very detailed control over their resources. This is not a feature of RBAC as it imposes controls over a group. Also often people occupy multiple roles in an application.

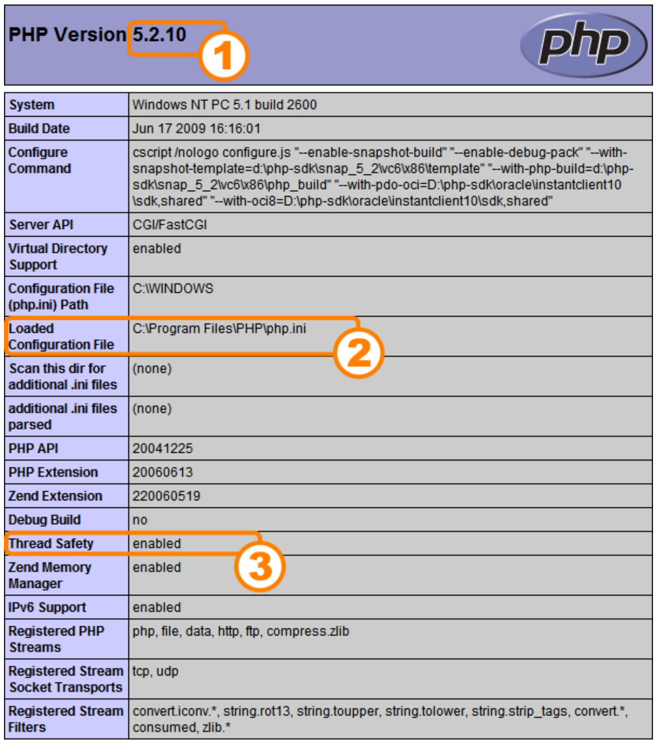
For e.g. take a bank application. Many people fill in the role for cashier in their absence even if they are clerks. A slightly modified version of RBAC is needed to accomplish this. Also performance and implementation issues are there to be considered. Currently there is no secure implementation that gives the user centralized authorization based on RBAC which is also compliant with NIST standards.

Implementation: This is a ready component based on NIST standards, used for handling of authorization, created by my mentor, Mr. Abbas Naderi. The official website for this is here. Multiple roles problem mentioned above is dealt in this implementation by providing temporary roles to users whenever needed after proper authorization. Note that this library would work with the authorization library discussed earlier in this document.

10. Secure Application Configuration and State

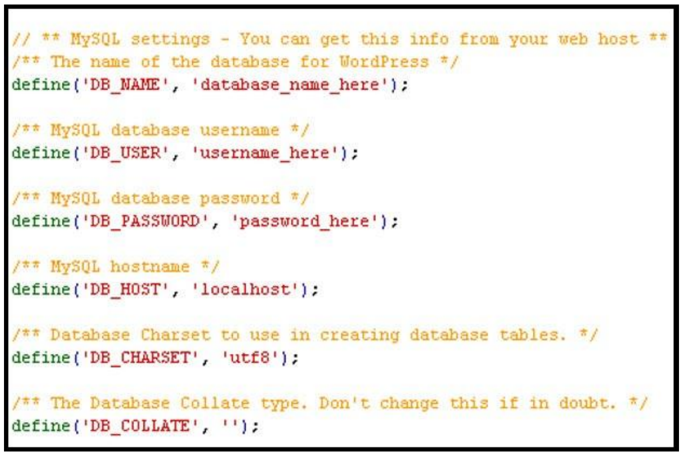
This library provides best practices for handling and management of application configuration securely. Application configuration examples are database credentials, versions, states, etc. Various attacks results from misconfiguration of data which is acknowledged by OWASP Top 10 many times in the past. Also in various cases leak of data such as version number, configuration states, and file disclosures reveal too much information to the attacker. This library would provide centralized means for most of the configuration management, and stores them encrypted and safe for developer's ease.

As can be seen that these are sensitive data and needs to be protected.



PHP Version 5.2.10

System	Windows NT PC 5.1 build 2600
Build Date	Jun 17 2009 16:16:01
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-snapshot-template=d:\php-sdk\snap_5_2\vc6\x86\template" "--with-php-build=d:\php-sdk\snap_5_2\vc6\x86\php_build" "--with-pdo-oci=D:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8=D:\php-sdk\oracle\instantclient10\sdk,shared"
Server API	CGI/FastCGI
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	C:\Program Files\PHP\php.ini
Scan this dir for additional .ini files	(none)
additional .ini files parsed	(none)
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	enabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	php, file, data, http, ftp, compress.zlib
Registered Stream Socket Transports	tcp, udp
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, zlib.*



```
// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define('DB_NAME', 'database_name_here');

/** MySQL database username */
define('DB_USER', 'username_here');

/** MySQL database password */
define('DB_PASSWORD', 'password_here');

/** MySQL hostname */
define('DB_HOST', 'localhost');

/** Database Charset to use in creating database tables. */
define('DB_CHARSET', 'utf8');

/** The Database Collate type. Don't change this if in doubt. */
define('DB_COLLATE', '');
```

11. PHP Error Handler Library

This would be a best practice error handler implemented for PHP that automatically detects what to do in case of an error, based on application state and error nature.

Handling of errors in PHP is tougher than other programming languages, because there are both Signal-Like errors (from the C world) and Exceptions (from high level world). Converting low level errors to Exceptions, and enable/disabling error handler with one function call is a tricky functionality usually implemented incorrectly.

Errors and exceptions have severity level. This library allows logging of errors in production environment and providing a general error interface to the user, so that user information is not disclosed to any attacker. Higher level errors are emailed to an administrator if set so.

Many 3rd party libraries (e.g PHPUnit) have built-in error handlers and require use of their error handler to function properly, thus our library needs to be able to shut-down itself when needed, and wake again afterwards. This functionality is also intended.

12. Extra Libraries

Note: Optional libraries. To be implemented if time permits.

i. **Secure SOAP and REST Library**
   This library handles web service security for PHP.
ii. **Secure Cache Library**
   This library handles secure cache for files and data. OWASP lists some attacks here based on cache where the attacker just browses through the cached files of browser and gets the information without having to pass the authentication mechanism. Many developers are not aware of this type of attack. This library will focus on various methods to reduce and control these types of attacks. e.g. use of "Cache-Control" and "Pragma" to control what pages must be cached and what not.
iii. **Security Scanner**
   The function of this library would be to warn the users of insecure practices of the features we have provided such as connecting to databases directly and using unsanitized data.
iv. **Secure Observer Pattern Library**
   Observer pattern is being used for extensibility in many applications today (e.g Wordpress). Unfortunately this has many side-effects and a secure library needs to handle them.
v. **Secure i18n Library**
   Internationalization is the process of making an application usable to different locales and languages. Nowadays almost all applications require this to reach a greater volume of customers. Unfortunately current practices of i18n are very obsolete and insecure (e.g gettext) and many security flaws are present in them.

References

- OWASP TOP 10
- OWASP PHP Security Cheat Sheat
- Worst Passwords
  - http://gcn.com/articles/2012/10/23/25-worst-passwords-2012.aspx
  - http://www.passworddragon.com/avoid-common-passwords
- Password Length Complexity
  - https://www.owasp.org/index.php/Password_length_%26_complexity
- LinkedIn Attack
- PHP RBAC
- Browser Cache

Timeline

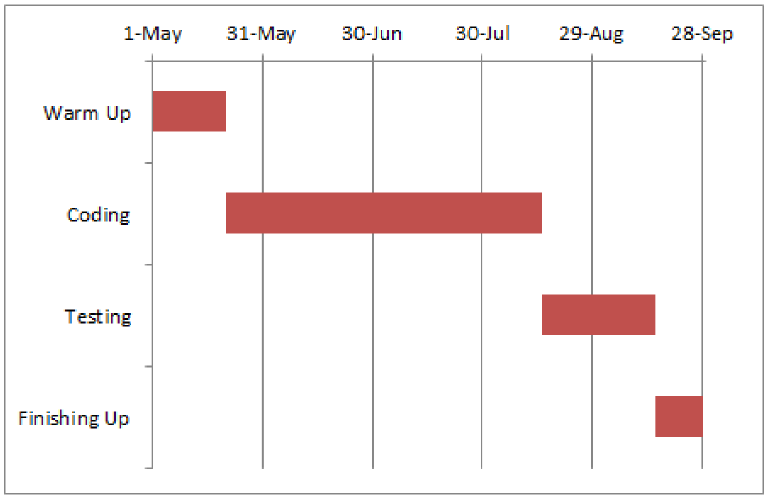
Note: I have planned to devote 70 hrs per week (10 hrs per day) on this project.

May 1-May 20 (Warm-Up): In this phase I plan to get concrete understanding of all the requirements and have detailed discussion for each of the library with my mentor. I will read existing documents and research papers suggested by him and will prepare sub-deadlines to check out progress in real-time. I will also start coding the basic libraries in this period.

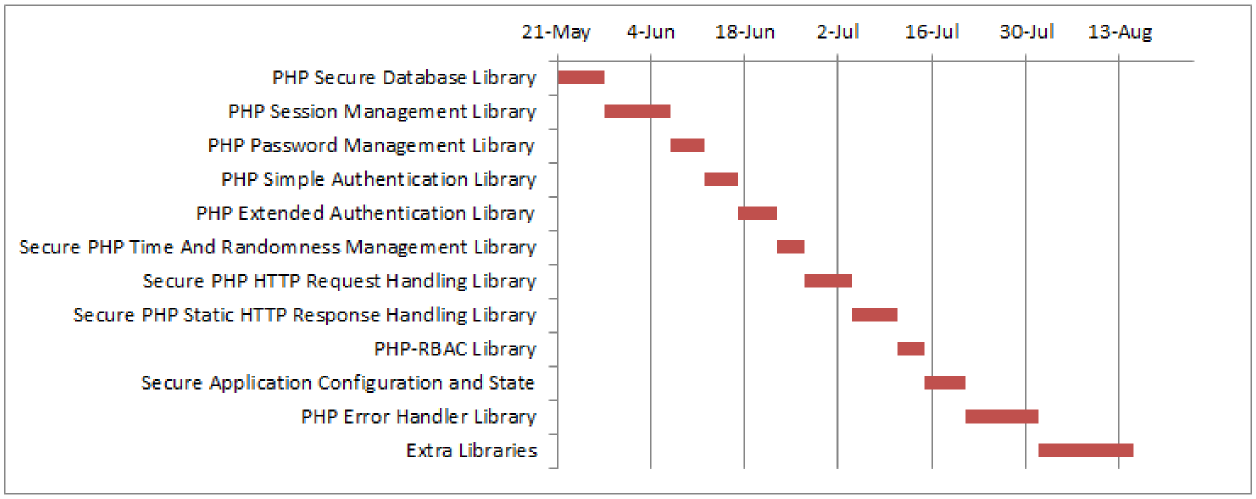
May 21-August 14 (Coding): In this phase I plan to code most of the libraries.

August 15-September 14 (Testing): In this phase I plan to test each of the library extensively with my mentor. We would try different attacks to circumvent the security and will fix bugs. I also plan to include my professors and co-workers to test these libraries for any bugs. In this phase I would also try to couple all the libraries into a framework.

September 15-September 27 (Finishing-Up): In this phase I plan to complete all the documentation and have future plans to extend the framework. Finally codes will be submitted on 27th of September.



(**Fig:** Timeline for overall project)



(**Fig:** Timeline based on each library implementation)

About Me

Introduction

My name is Rahul Chaudhary and I am a Master's student at "University of Pittsburgh". My field of specialty here is "Information Security". This is the first time I am participating in GSOC.
I see myself as a fit candidate mainly because of my enthusiasm towards this project. **I have taken my time off from all other activities such as extra classes to participate in GSOC; thus I have lots of free time to devote in this project.** Also my knowledge in security is fair which could be used extensively here.

Previous Projects

I have been involved in many web-application projects and almost all of them involved PHP. In my undergraduate career I did a project for my state govt. where I was told to develop an interface to monitor all the cyber-cafés in my state. During that period I was also involved in one IBM summer project on "Smart Inventory Management System" hosted by my college.
Lately I have been involved in making a Facebook application for collecting privacy information from users and displaying them in a comprehensible way. This project was for research purpose funded by my school in "University of Pittsburgh". Currently I am building a social-networking site to support abused victims. This project also is funded by my school.

Motivation

This is my first chance to get involved this deep in an open-source community and I am very excited to contribute my part in making this world a better place. Until now I felt I was too immature to join open-source community. But GSOC has given me the perfect opportunity to get involved in these groups and to exercise my knowledge. Security is my passion and I have learned most of the things in this field because of OWASP (it's like a security Wikipedia to me). Getting involved with this organization and to contribute to the open-source community, in itself is a big motivation. I look forward to learn new things on my way.

3rd Party Involvement

Following persons **POSSIBLY COULD** provide feedbacks on my work from time-to-time:

- Prof. Sidney Faber (sid@fabersl.com): He works in CERT (CMU) and also teaches "Security Management" in University of Pittsburgh. After discussing this project with him, he promised that he would try to involve other people from CERT to informally guide me on this project.

- Dr. James B. D. Joshi (jjoshi@mail.sis.pitt.edu): He works in University of Pittsburgh and is the head of security dept. of "School of Information Sciences".

Following persons **POSSIBLY COULD** participate in testing of our library once it has been created:

- PhD students of "School of Information Sciences - University of Pittsburgh". I am trying to include some of them for testing. If they agree, then the details of their works will be documented and provided with their names on it and not mine. The reason for them to involve is not to reduce my work load, but is to get a new perspective from other people because I believe that the more people who will assess my work, the more solid result I would be able to give. And at the end of the day, only that matters.

Contact Information

Student Info

Name: Rahul Chaudhary
E-Mail: rac130@pitt.edu (If needed, I will disclose my other contact information).
Phone: +1 412 519 9634
Emergency Contact Person: Dr. Amar Kumar Chaudhary (Father) (+91 9931132907)
Location: 5835 Alderson St., Pittsburgh, PA, USA (15217)

Mentor Info

Name: Abbas Naderi
E-Mail: abiusx@owasp.org
Web-Page: https://www.owasp.org/index.php/User:Abbas_Naderi#Contribution
Mentor Information: Mr. Abbas Naderi is chapter leader of Iran in OWASP and has participated in OWASP Projects for more than 5 years. Major projects where he contributed are ASVS, ESAPI, WebGoat etc. He is also the project leader for implementing RBAC in PHP (http://phprbac.net/) and is leading many projects this year in GSOC.