PHP SECURITY AND CONCERNS

A. Yeshvini, Kanika Arora
Btech, Dronacharya College Of Engineering ,Gurgaon, India

Abstract- **About 30% of all vulnerabilities listed on the Vulnerability information are coupled to PHP vulnerabilities are caused principally by not following best-practice programming rules. Technical security flaws of the language itself or of its core libraries don't seem to be frequent (23 in 2008, regarding one hundred and twenty fifth of the total). Recognizing that it is usual that programmers build mistakes, some of the languages embrace taint checking to mechanically notice the shortage of input validation that induces several problems. Such a feature is being developed for PHP, however its inclusion during a unleash has been rejected many times within the past. There are advanced protection patches such as Suhosin and Hardening-Patch, especially designed for web hosting environments. Some of the vulnerabilities are iatrogenic by improper PHP's runtime configuration. as an example, failing to disable PHP execution for the directory wherever uploaded pictures are hold on, may end up in execution of malicious PHP embedded inside uploaded pictures. Another standare example is departure enabled the dynamic loading of PHP extensions, during a shared hosting atmosphere.In this paper we've got represented all the aspects and usage of php together with its linking ability to the remote sites and servers and conjointly in networking and security.**

1. INTRODUCTION

This document describes the functionality and statistics related to php language and security aspects of it.

PHP is a powerful language and the interpreter, whether included in a web server as a module or executed as a separate CGI binary, is able to access files, execute commands and open network connections on the server. These properties make anything run on a web server insecure by default. PHP is designed specifically to be a more secure language for writing CGI programs than Perl or C, and with correct selection of compile-time and runtime configuration options, and proper coding practices, it can give you exactly the combination of freedom and security you need.

As there are many different ways of utilizing PHP, there are many configuration options controlling its behaviour. A large selection of options guarantees you can use PHP for a lot of purposes, but it also means there are combinations of these options and server configurations that result in an insecure setup.

The configuration flexibility of PHP is equally rivalled by the code flexibility. PHP are often accustomedbuild complete server applications, with all the facility of a shell user, or it are often used for easyserver-side includes with very little risk in a very tightly controlled setting. however you build thatsetting, and the way secure it's, is essentially up to the PHP developer.

This chapter starts with some general security advice, explains the different configuration option combinations and the situations they can be safely used, and describes different considerations in coding for different levels of security

General considerations

A completely secure system is a virtual impossibility, so an approach often used in the security profession is one of balancing risk and usability. If every variable submitted by a user required two forms of biometric validation (such as a retinal scan and a fingerprint), you would have an extremely high level of accountability. It would also take half an hour to fill out a fairly complex form, which would tend to encourage users to find ways of bypassing the security.

The best security is often unobtrusive enough to suit the requirements without the user being prevented from accomplishing their work, or over-burdening the code author with excessive complexity. Indeed, some security attacks are merely exploits of this kind of overly built security, which tends to erode over time.

A phrase worth remembering: A system is only as good as the weakest link in a chain. If all transactions are heavily logged based on time, location, transaction type, etc. but the user is only verified based on a single cookie, the validity of tying the users to the transaction log is severely weakened.

When testing, keep in mind that you will not be able to test all possibilities for even the simplest of pages. The input you may expect will be completely unrelated to the input given by a disgruntled employee, a cracker with months of time on their hands, or a housecat walking across the keyboard. This is why it's best to look at the code from a logical perspective, to discern where unexpected data can be introduced, and then follow how it is modified, reduced, or amplified.

The Internet is filled with people trying to make a name for themselves by breaking your code, crashing your site, posting inappropriate content, and otherwise making your day interesting. It doesn't matter if you have a small or large site, you are a target by simply being online, by having a server that can be connected to. Many cracking programs do not discern by size, they simply trawl massive IP blocks looking for victims. Try not to become one.

II. TYPES OF SECURITY UNDERTAKEN

Making the system, files ANd services is nowadays a chiefly followed facet whereas e b eneathparturition the standards for development beneath an atmosphere. Php on the opposite hand employs varied commands and languages with it to prove it for the protection functions. though theoperating conditi ons deployed and standards beside the definitions varies it's has the power to indicate its practicality for the sectors as listed below

.2.1File System Security

PHP is subject to the security built into most server systems with respect to permissions on a file and directory basis. This allows you to control which files in the filesystem may be read. Care should be taken with any files which are world readable to ensure that they are safe for reading by all users who have access to that filesystem.

Since PHP was designed to permit user level access to the filesystem, it's entirely attainable to jot down a PHP script that may permit you to scan system files like /etc/passwd, modify your LAN connections, sendlarge printer jobs out, etc. This has some obvious implications, therein you would like to confirm that the files that you simply scan from and write to area unit the acceptable ones

.2.2Database Security

Nowadays,databases are cardinal parts of any net based mostly application by sanctionativewebsites tosupply variable dyna mic content. Since terribly sensitive or secret data will be keepduring a info, you must powerfully take into account protective databases.

To retrieve or to store any data you wish to attach to the info, send a legitimate question, fetch the result, and shut the affiliation. Nowadays,the usually used search language during,thtis interaction is that the Structured Query language (SQL). See however associate offender will tamper withassociate SQL question.As you can surmise, PHP cannot protect your database by itself. The following sections aim to be an introduction into the very basics of how to access and manipulate databases within PHP scripts.

Keep in mind this simple rule: defense in depth. The more places you take action to increase the protection of your database, the less probability of an attacker succeeding in exposing or abusing any stored information. Good design of the database schema and the application deals with your greatest fears.

III. SETTING SECURE PHP AND CONFIGURATION SETTINGS

PHP code can be embedded in your Web pages along with HTML code. When your Web server receives a request for a page, the page is first given to the PHP handler. The PHP handler outputs HTML code without modification and executes any PHP commands. Any HTML code generated by the PHP commands is also output. This results in a Web page with content that has been customized on the server before being sent to the requestor.

The capabilities of PHP also make it a potential security risk because data is actively fetched, received, and processed from anywhere on the Internet. Attackers may attempt to send in malicious data and scripts and trick your server into fetching malicious scripts and running them. Attackers may also attempt to read and write files on your server to take control of the Web site and use it for their own purposes.

You can configure PHP settings to tighten the security of a PHP installation and help protect the Web site from malicious attacks. The Php.ini file specifies the configuration settings PHP uses when it is running on your Web site. The Php.ini file determines what things PHP scripts are allowed to do and what the scripts are prohibited from doing. Table 1 summarizes settings that affect security. More detailed explanations of the setting follow.

Setting	**Description**

allow_url_fopen=Off
allow_url_include=Off
	Disable remote URLs (which may cause code injection vulnerabilities) for file handling functions.

register_globals=Off	Disable register_globals.
open_basedir="c:\inetpub\"	Restrict where PHP processes can read and write on a file system.

safe_mode=Off
safe_mode_gid=Off
	Disable safe mode.

max_execution_time=30
max_input_time=60
	Limit script execution time.

memory_limit=16M
upload_max_filesize=2M
post_max_size=8M
max_input_nesting_levels=64
	Limit memory usage and file sizes.

display_errors=Off
log_errors=On
error_log="C:\path\of\your\choice"
	Configure error messages and logging.

fastcgi.logging=0	Internet info Services (IIS) Fast CGI module can fail the request once PHP sends any information on stderr by exploitation FastCGI protocol. Disabling FastCGI work can forestall PHP from causing error info over stderr, and generating five hundred response codes for the shopper.
expose_php=Off	Hide presence of PHP.

IV. ERROR REPORTING

With PHP security, there are two sides to error reporting. One is beneficial to increasing security, the other is detrimental.

A standard attack tactic involves profiling a system by feeding it improper data, and checking for the kinds, and contexts, of the errors which are returned. This allows the system cracker to probe for information about the server, to determine possible weaknesses

The PHP errors which are normally returned can be quite helpful to a developer who is trying to debug a script, indicating such things as the function or file that failed, the PHP file it failed in, and the line number which the failure occurred in. This is all information that can be exploited. It is not uncommon for a php developer to use show_source(), highlight_string(), or highlight_file() as a debugging measure, but in a live site, this can expose hidden variables, unchecked syntax, and other dangerous information. Especially dangerous is running code from known sources with built-in debugging handlers, or using common debugging techniques.

4.1 Issues

If you are a "connected" developer, you are probably aware of the major vulnerabilities found in Ruby on Rails recently. To be fair, we've also found some serious issues in the Symfony code during the last few months.

As security management ought to be a high most priority for United States of America and our customers, security management additionally| a really vital topic on behalf of me as a result of Symfony is quickly growing in quality end user comes and ASCII additional exposure also means that additional interest from the "bad guys"

One of the goal of good security issues management is **transparency**. That's why the Symfony project has a simple way of reporting security issues (via the security [at] symfony.com email address), an easily accessible list of security advisories , and a well defined blog post template to announce security issues. Recently, we have also enforced the need to have a CVE identifier for all security issues to help the broader community to be aware of Symfony security issues.

V. CONTROLLING PHP

In general, security by obscurity is one of the weakest forms of security. But in some cases, every little bit of extra security is desirable.

A few simple techniques can help to hide PHP, possibly slowing down an attacker who is attempting to discover weaknesses in your system. By setting expose_php to ***off*** in your *php.ini* file, you reduce the amount of information available to them.

Another tactic is to configure web servers such as apache to parse different filetypes through PHP, either with an *.htaccess* directive, or in the apache configuration file itself. You can then use misleading file extensions.

VI. CONCLUSION

This research paper underlines the requirement and various features of php language and its significance in security. In the past PHP open source principles have put greater overhead on individual developer choices. Unlike Java, every developer had the possibility to do as they pleased since there was little language and systems standardization. Over the past 18 months this gap has been minimized by the introduction of a set of best practices and standardizations and will continue this orientation in future with the growth of PHP frameworks. It had work for the development and welfare of the IT world. The ways of it implementation in services and remote applications being developed at a large scale thus increasing the demand of developers made it more strong with the problems and frequent queries. We have also listed its large scale implementations along with the issues overcome with time. Also we tried to give a good overview over the different security hardening features an attacker will have to deal with once he succeeded in executing arbitrary PHP code. However this is only the beginning of our research, because valid countermeasures against this attack have yet to be developed. The first ideas to stop this kind of attack were all not sufficient to really solve the problem. Delaying the execution of user-space error handlers until internal functions have ended does only solve a part of the problem, the same is true for removing the calltime-passby-reference feature once and for all. There are still exploitation path that are not covered by this, like the usort() vulnerability. Fixing the PHP code to not have interruption vulnerabilities is also no short time solution, because many areas of the code would have to be checked. Aside from that there are only hacks that try to hinder the described exploitation paths by changing structures

REFRENCES

[1] PHP Documentation Team, ”PHP: Safe Mode - Manual”, PHP Documentation, http://www.php.net/manual/en/features.safe-mode.php

[2] PHP Documentation Team, ”PHP: Security and Safe Mode - Manual”, PHP Documentation, http://de.php.net/manual/en/ini.sect.safe-mode.php#ini.open-basedir

[3] Secunia, ”Secunia Advisory Search for PHP + Safe Mode”, Secunia Advisories,

http://secunia.com/advisories/search/?search=PHP+safe+mode
[4] Secunia, ”Secunia Advisory Search for PHP + Safe Mode + Curl”, Secunia Advisories, http://secunia.com/advisories/search/?search=PHP+safe+mode+curl
[5] J. Dahse, ”Safe mode bypass”, http://bugs.php.net/bug.php?id=45997
[6] M. Arciemowicz, ”tempnam() open basedir bypass PHP 4.4.2 and 5.1.2”, http://securityreason.com/achievement securityalert/36
[7] M. Arciemowicz, ”PHP 5.2.3, htaccess safemode and open basedir Bypass”, http://securityreason.com/achievement securityalert/9
[8] S. Esser, ”Month of PHP Bugs”, http://www.php-security.org
[9] S. Esser, ”PHP memory limit remote vulnerability”, e-matters Advisory 11/2004, http://www.hardened-php.net/advisory em112004.100.html
[10] S. Esser, ”PHP register globals Activation Vulnerability in parse str()”, Hardened-PHP Advisory 19/2005, http://www.hardened-php.net/advisory 192005.78.html
[11] S. Esser, ”What is Suhosin?”, Suhosin Website, http://www.suhosin.org
[12] H. Etoh, ”GCC extension for protecting applications from stack-smashing attacks”, http://www.trl.ibm.com/projects/security/ssp/
[13] U. Drepper, ”Security Enhancements in Redhat Enterprise Linux (beside SELinux)”, 12/2005, http://people.redhat.com/drepper/nonselsec.pdf
[14] S. Esser, ”E-mail introducing the safe unlink concept”, 12/2003, http://archives.neohapsis.com/archives/bugtraq/2003-12/0014.html
[15] huku, ”Yet another free() exploitation technique”, Issue 66, Article 6, http://www.phrack.org/issues.html?issue=66&id=6
[16] blackngel, ”MALLOC DES-MALEFICARUM”, Issue 66, Article 10, http://www.phrack.org/issues.html?issue=66&id=10
[17] N. Provos, ”Systrace - Interactive Police Generation for System Calls”, http://www.citi.umich.edu/u/provos/systrace/
[18] Various Authors, ”AppArmor Detail”, 2009, http://en.opensuse.org/AppArmor Detail
27
[19] B. Spengler, ”PaX: The Guaranteed End of Arbitrary Code Execution”, 2003, http://www.grsecurity.net/PaX-presentation files/frame.htm
[20] The PaX Team, ”PaX Documentation”, 2003, http://pax.grsecurity.net/docs/pax.txt
[21] B. Spengler, ”Grsecurity”, http://www.grsecurity.net
[22] M. T. Jones, ”Anatomy of Security-Enhanced Linux (SELinux) - Architecture and Implementation”, 2008, http://www.ibm.com/developerworks/linux/library/l-selinux/
[23] The PaX Team, ”PaX Documentation: ASLR”, 2003, http://pax.grsecurity.net/docs/aslr.txt
[24] The PaX Team, ”PaX Documentation: NOEXEC”, 2003, http://pax.grsecurity.net/docs/noexec.txt
[25] Microsoft, ”A detailed description of the Data Execution Prevention (DEP) feature in Windows XP Service Pack 2, Windows XP Tablet PC Edition 2005, and Windows Server 2003”,
2006, http://support.microsoft.com/kb/875352
[26] H. Shacham, ”The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86)”, In Proceedings of CCS 2007, pages 552561, ACM Press, Oct. 2007,
http://cseweb.ucsd.edu/ hovav/dist/geometry.pdf
[27] The PaX Team, ”PaX Documentation: MPROTECT”, 2003, http://pax.grsecurity.net/docs/mprotect.txt
[28] S. Krahmer, ”x86-64 buffer overow exploits and the borrowed code chunks exploitation technique”, 2005, http://www.suse.de/ krahmer/no-nx.pdf