应用开发安全指南

目录

1.1基本代码安全要求

1.1.1 输入验证

对函数入口参数的合法性和准确性进行检查，具体如下：

- 在B/S环境下，应进行服务端的验证而不仅仅是客户端的验证（例如基于

Javascript的验证）。通过在客户端和服务器之间放置一个代理服务器，可以很容易绕过客户端验证。有了代理服务器，攻击者可以在数据被客户端“验证”后修改数据（与“中间人”攻击类似）。

- 在实际的校验中，输入校验首先定义一个有效（可接受）的字符集，然后检查每个数据的字符是否在有效范围内。如果输入中包含无效的字符，应用程序应该返回错误页面并说明输入中包含无效字符。这样进行验证的原因是定义无效的字符集比较困难，并且一些不应该有效的字符通常不会被指出。
- 另外，边界检查（例如字符串的最大长度）应该在字符有效性检查以前进行，边界分析可以防止大多数缓冲区溢出漏洞。
- 从环境变量获得的数据也需要进行验证，同时避免在环境变量中存放敏感数据（例如密码）。

1.1.2 SQL 语句

如果应用程序需要连接后端数据库，使用存储过程而不能在代码中使用 SQL 语句，使用程序以外的嵌入在代码中的 SQL 语句调用特别危险，难以防止攻击者使用输入域或者配置文件（由应用程序载入）来执行嵌入式的 SQL 攻击。当然，输入验证有助于缓解这种风险。

1.1.3 注释代码

当应用程序在实际环境中开始应用时，应该删除所有的注释代码。注释代码是用来调试或者测试的，它们不是最终应用程序的一部分。无论如何应该在实际的环境中删除它们以避免意外的执行（一般注释标识被删除后就无法激活休眠的代码，但还是存在可能性的，所以强烈建议执行这项工作）。

1.1.4 错误消息

所有为用户显示的错误信息都不应该暴露任何关于系统、网络或应用程序的敏感信息。如果可能，应使用包含编号的一般的错误信息，这种信

息不返回给访问用户，只返回 404 错误，如“发生了错误（代码 1234），请您与系统维护部门联系。

1.1.5 URL 内容

对于 Web 应用，不能在 URL 上暴露任何重要信息，例如密码、服务器名称、IP 地址或者文件系统路径（暴露了 Web 服务器的目录结构），这些信息可以在攻击时被使用。

1.1.6 设置 PATH 变量

设置 PATH 为一个已知的值，而不是仅仅使用启动时的缺省值。攻击者可以在攻击应用程序时使用 PATH 变量，例如试图执行一个任意的程序，这些可以应用于大多数其他的语言。

1.1.7 其他要求

1、 禁止使用未经授权和验证的代码。

2、 使用第三方代码，应对代码安全性进行评估和测试。

3、 测试用的“后门”，应在发布版中去除。

4、 规范代码的格式。

5、 规范变量、函数的命名；

6、 规范程序的书写格式，确保程序的易读性。

7、 对代码进行版本控制，确保代码的可用性。

8、 防止程序员非授权修改代码

9、 对代码的访问权限进行严格的权限控制；

10、 禁止在程序中添加隐藏“恶意”的代码，防止与应用系统相关的程序员对系统的非授权修改。

11、 应用系统不应在程序或进程中固化账号和口令。

12、 系统应具备对口令猜测的防范机制和监控手段。

13、 避免应用程序以错误的顺序运行，或者防止出现故障时，后续程序以不正

常的流程运行。

14、 采用正确的故障恢复程序，确保正确处理数据。

15、 采取会话控制或批次控制，确保更新前后数据文件的一致性，例如：检查操作前后文件打开和关闭的数目是否一致。

16、 检查执行操作前后对象的差额是否正常，如：句柄处理，堆栈等系统资源的占用与释放等。

17、 严格验证系统生成的数据。

18、 在网络传输过程中检查下载/上传的数据或软件的完整性。

19、 检查文件与记录是否被篡改。例如通过计算哈希值（HASH）进行对比。

20、 禁止私自讲代码上传到互联网。

1.2Web 编程安全基本要求

1.2.1 输入检查安全

1、 限制用户输入 HTML 和 Script(JavaScript、VBScript)代码。输入恶意 HTML 或 Script（JavaScript、VBScript）代码可能会对其他浏览者造成混淆、欺骗或恶意破坏的结果。

2、 检查用户输入数据的长度。输入超出限定长度的数据，可能造成服务器端程序溢出。

3、 防止用户输入特殊字符改变 SQL 语义。输入含特殊字符的字串，篡改 SQL 语句的语义，可能造成 SQL 查询执行不该执行的操作，以此绕过身份认证获取非法权限、甚至对数据进行破坏。

4、 限制用户能够访问的最顶层目录。编写对服务器端文件、目录操作的程序时应该注意限定此类程序能够访问的最顶层目录，防止用户构造输入字串借助程序功能访问服务器关键文件导致泄漏服务器敏感信息。

5、 对所有类型的用户输入都要做检查，并严格限定什么是合法的用户输入，限定一个合法输入的范围，同时过滤有可能造成危险的特殊字符。

6、 对不可信任域发送到可信任域的数据一定要进行检查。

7、 尽可能在服务器端完成用户输入检查，不能轻易相信客户端脚本的检查结果。虽然客户端的 Script 脚本能完成一部分的用户输入检查功能，但这种检查的结果是不可信任的，攻击者可以自己制作表单程序绕过客户端脚本验证，将非法数据提交到服务器。
8、 在输入变为输出时，也要对特殊字符做检查和转换。
9、 判断输入的 id 为 list 的情况：如 http://xx.xx.xx/id[]=1

1.2.2 敏感数据的存放和传递安全

1、 敏感数据不能存放在 Web 页中。
2、 不能把敏感的数据存储在 cookie、隐藏字段或者潜在地可能会被用户修改的地方。
3、 客户端向服务器端提交敏感数据应该经过加密（例如使用 SSL），尽量不能明文传输。
4、 密码等敏感信息存放在数据库中应该加密，并采用健壮的加密算法。
5、 防止数据库被攻破后泄漏用户密码。
6、 敏感数据需要脱敏显示。

1.2.3 缓冲区溢出安全

1、 所有的输入都必须进行正确的有效性检测。
2、 必须保证数组没有越界，增加数组操作函数的边界检查。
3、 安全地使用字符串处理函数，慎用有安全隐患的字符串处理函数
4、 使用 Format 字符串的时候特别注意 Unicode 和 ANSI 的大小不一致的情况。
5、 注意字符串结束符的保护。
6、 仔细研究库函数内部的缓冲区分配，明确其限制。不能使用 realpath() 等函数，如果功能需要必须使用时，一定要检查试图规范化的路径的长度，确保其不长于 MAXPATHLEN。
7、 时刻进行边界检查。建议使用一些检查工具：Purify、Stackguard 等检查代码，保证没有缓冲区溢出的问题。

1.2.4 格式化字符串安全

1、 使用固定的格式化字符串，或者来自可信源的格式化字符串。
2、 要检查并限定 locale 的请求为合法值。
3、 不能将用户输入直接作为格式化字符传给格式化函数。

1.2.5 整数溢出安全

1、 对于涉及到内存分配大小的计算，要进行仔细检查，确保计算不会产生溢出。
2、 对于涉及到数组索引的计算，要进行仔细检查，确保计算不会产生溢出。
3、 要使用无符号整数表示数组偏移和内存分配大小。

1.2.6 SQL 注入代码安全

1、 要检查输入的有效性和可信度。
2、 要使用参数化的查询、占位符、或者参数绑定来构造 SQL 语句。
3、 要在程序之外存储数据库的连接信息，比如经过保护的配置文件或者 Windows 注册表。
4、 即使使用的是存储过程，也不能使用字符串连接来构造 SQL 语句。
5、 不能在存储过程内部使用字符串连接来构造 SQL 语句。
6、 不能在存储过程内部执行不可信的参数。
7、 不能简单地双写单引号或者双引号。
8、 不能使用高权限账号连接数据库，比如 sa 或者 root。
9、 不能在程序或者连接字符串中存储登录口令。
10、 不能在 Web 根目录下存储数据库配置信息。
11、 应从数据库中删除对所有用户自定义表的访问权限，同时只对存储过程授权，然后使用存储过程以及参数化的查询来构造查询字符串。

1.2.7 命令注入代码安全

1、 在输入命令传递给命令处理程序之前要进行验证。

2、 如果输入验证失败，要安全地处理失败信息。
3、 不能向任何命令解释器传递未验证的输入信息，即使这些输入仅仅是数据信息。
4、 避免使用正则表达式来进行输入验证，应手工去写一些简单而又清晰的验证代码。

1.2.8 异常处理代码安全

1、 要检测每个安全相关函数的返回值。
2、 对于每一个更改用户设定或者及其设定的函数，都要检查其返回值。
3、 要有从错误条件中进行恢复的考虑，避免拒绝服务攻击。
4、 不能一次性处理所有的异常，要将异常情况进行分类处理，避免在异常处理代码中的漏洞发生。

1.2.9 跨站脚本代码安全

1、 要对所有基于 Web 的输入进行输入验证和可信度验证。
2、 在没有验证合法性之前，不能对基于 Web 的输入进行回显。
3、 不能在 cookie 中存储敏感数据。

1.2.10 保护网络流量的代码安全

1、 要使用强大的初始认证机制。
2、 对应用程序所产生的所有网络流量都要执行过程中消息认证。
3、 尽可能使用 SSL/TLS 进行网络加密传输。

1.2.11 应用中的弱口令代码安全

1、 确保口令在网络上认证时不被窃听。
2、 要在登录失败时给出错误提示，并记录失败口令尝试。
3、 尽可能使用基于 hash 强壮的单向加密函数进行口令存储。
4、 为用户更改口令提供安全的机制。

5、 不得使用默认账号和默认口令，若使用，必须在首次登录后进行修改。

6、 不得在程序、后台存储明文的口令。

7、 口令要有一定的强度，应当满足系统的账号口令策略要求。

1.2.12 SOCKET 网络编程安全基本要求

1、 在 socket 函数调用时，明确参数中绑定的端口、IP 地址和网卡接口。Windows 环境下，在遇到多个网卡的情况时，需要通过注册表来获得网卡接口和 IP 地址的信息，包括 WindowsNT 和 windows2008。

2、 判断连接的合法身份。即，为防止恶意的连接以及可能是无效的连接，建议在 socket 连接期间，判断连接的对端是否是合法的真正的连接。

3、 对于 UDP 连接，可以获得连接对方的 IP 地址和端口，从而可以判断对方的有效性和合法性；对于 TCP 连接，由于每次连接需要三次握手，而且还有超时机制，存在两种方式来控制。

4、 对于 TCP 连接，需要尽量在三次握手完成前完成判断，同时防止端口扫描的攻击。

5、 尽可能确保 socket 应用能通过合理设置的防火墙。

6、 在可能的情况下，尽量减少 socket 连接数目。

7、 尽量不能使用回拨的技术。

8、 尽量采用有连接状态的协议，例如 TCP 协议。由于防火墙一般采取禁止一切的策略，对于 UDP 协议比较难以设置。

9、 在一个应用程序中，尽量使用同一种协议，不能使用多种协议。

10、 尽量将客户端和服务器端的端口做成可以配置，不能硬编码在程序中。

1.3JAVA 安全开发要求

JAVA 语言安全规范参考 OWASPTOP10 要求，本指南列举了常见的 JAVA 开发安全要求。

1.3.1 防范跨站脚本（XSS）

跨站脚本是最普遍的 Web 应用安全漏洞。当应用程序在发送给浏览器的页面中包含用户提供的数据，但没有经过适当验证或转译，就容易导致跨站脚本漏洞。

攻击者能在受害者浏览器中执行脚本以劫持用户会话、危害网站、插入恶意内容和重定向用户等。

已知三种著名跨站漏洞是：1）存储式；2）反射式；3）基于 DOM。

反射式跨站脚本通过测试或代码分析很容易找到。

防范措施：

1、验证输入

检查每个输入的有效性，主要检查输入类型和数据的长度。

2、编码输出

对验证输入的另一面就是编码输出。编码输出是指确保字符被视为数据，而不是作为 HTML 元字符被浏览器解析。这些技术定义一些特殊的“转义”字符，没有正确转义的数据它仍然会在浏览器中正确解析。编码输出只是让浏览器知道数据是不是要被解析，达到攻击无法实现的目的。需要编码的部分：HTML 实体、HTML 属性、JavaScript、CSS、URL。

1.3.2 防范 SQL 注入

简单来说，注入往往是应用程序缺少对输入进行安全性检查所引起的，攻击者把一些包含指令的数据发送给解释器，解释器把收到的数据转换成指令执行。注入漏洞十分普遍，通常能在 SQL 查询、LDAP 查询、Xpath 查询、OS 命令、程序参数等中出现。

注入能导致数据丢失或数据破坏、缺乏可审计性或是拒绝服务，注入漏洞有时甚至能导致完全接管主机。

SQL 注入包含了 SQL 注入、XPATH 注入、LDAP 注入、OS 命令注入等。

1.3.3 防范恶意文件执行

恶意文件执行是一种能够威胁任何网站形式的漏洞，只要攻击者在具有引入（include）功能程式的参数中修改参数内容，Web 服务器便会引入恶意程序从而受到恶意文件执行漏洞攻击。

攻击者可利用恶意文件执行漏洞进行攻击取得 Web 服务器控制权，进行不法利益或获取经济利益。

1.3.4 不安全的直接对象引用

所谓“不安全的对象直接引用”，即 Insecure direct object references，意指一个已经授权的用户，通过更改访问时的一个参数，从而访问到原本其并没有得到授权的对象。Web 应用往往在生成 Web 页面时会用它的真实名字，且并不会对所有的目标对象访问时检查用户权限，所以这就造成了不安全的对象直接引用的漏洞。以下是不安全的对象直接引用示例：

- 攻击者发现他自己的参数是 6065，即?acct=6065；
- 他可以直接更改参数为 6066，即?acct=6066；
- 这样他就可以直接看到 6066 用户的账户信息了；
- 这种漏洞能损害参数所引用的所有数据。除非名字空间很稀疏，否则攻击者很容易访问该类型的所有数据。

1.3.5 防范跨站请求伪造

跨站请求伪造，也被称成为“one-click attack”或者 session riding，通常缩写为 CSRF 或者 XSRF，是一种对网站的恶意利用。它与 XSS 非常不同，并且攻击方式几乎相左。XSS 利用站点内的信任用户，而 CSRF 则通过伪装来自受信任用户的请求来利用受信任的网站。与 XSS 攻击相比，CSRF 攻击往往不太流行（因此对其进行防范的资源也相当稀少）和难以防范，所以被认为比 XSS 更具危险性。

攻击者能让受害用户修改可以修改的任何数据，或者是执行允许使用的任何功能。

1.3.6 信息泄露和错误处理不当

应用程序常常产生错误信息并显示给使用者。很多时候，这些错误信息非常有用，因为它们揭示实施细则或有用的开发信息。泄露太多的细节（如错误堆栈跟踪信息、SQL 语句等等）；

登录失败后，通知用户是否用户 ID 或密码出错——登录失败可能是由于 ID 或密码错误造成的。这为一个对关键资产发动蛮力攻击的攻击者提供重要信息。

1.3.7 残缺的认证和会话管理

与认证和会话管理相关的应用程序功能往往得不到正确实施，这就导致攻击者破坏密码、密钥、会话令牌或利用实施漏洞冒充其他用户身份。

这些漏洞可能导致部分甚至全部账号遭受攻击。一旦攻击成功，攻击者能执行合法用户的任何操作，因此特权账号会造成更大的破坏。

编程要求：

- 使用内置的会话管理功能；
- 通过认证的问候；
- 使用单一的入口点；
- 确保在一开始登录 SSL 保护的网页；
- 获取注销的权利；
- 添加超时；
- 确保你使用的是安全相关的功能；
- 使用强大的认证；
- 不进行默认身份验证。

1.3.8 不安全的加密存储

保护敏感数据已经成为网络应用的最重要的组成部分，加密的敏感数据已是非常常见安全保护手段。不加密的应用程序、设计不当或者使用不恰当的密码技术等可能导致披露敏感数据。

- 攻击者能够取得或是篡改机密的或是私有的信息；

- 攻击者通过机密秘密的窃取从而进行进一步的攻击；
- 造成企业形象破损，用户满意度下降，甚至面临法律诉讼等。编程要求：
- 验证你的结构；
- 识别所有的敏感数据；
- 识别敏感数据存放的所有位置；
- 确保所应用的威胁模型能够应付这些攻击；
- 使用加密手段来应对威胁；
- 使用一定的机制来进行保护
- 文件加密；
- 数据库加密；
- 数据元素加密；
- 正确的使用这些机制；
- 使用标准的强算法；
- 合理的生成，分发和保护密钥；
- 准备密钥的变更；
- 验证实现方法；
- 确保所有的证书、密钥和密码都得到了安全的存放；
- 有一个安全的密钥分发和应急处理的方案。

1.3.9 不安全的通信

对于不加密的应用程序的网络信息传输，需要保护敏感的通信。加密（通常 SSL）必须用于所有身份验证的连接，特别是通过 Internet 访问的网页，以及后端的连接。否则，应用程序将暴露身份验证或会话令牌。

- 攻击者能够取得或是篡改机密的或是私有的信息；
- 攻击者通过这些秘密的窃取从而进行进一步的攻击；
- 造成企业形象破损，用户满意度下降，甚至面临法律诉讼等。
- 编程要求：
- 提供合理的保护机制；
- 对于敏感数据的传输，对所有连接都要使用 TLS；

- 在传输前对单个数据都要进行加密；（如 XML-Encryption）；
- 在传输前对信息进行签名；（如 XML-Signature）；
- 正确的使用这些机制；
- 使用标准的强算法；
- 合理管理密钥和证书；
- 在使用前验证 SSL 证书。

1.3.10 限制 URL 访问失效

这个漏洞事实上也是与认证相关的，与我们前面提到的不安全的直接对象引用也是类似的，不同在于这个漏洞是指系统已经对 URL 的访问做了限制，但这种限制却实际并没有生效。常见的错误是，我们在用户认证后只显示给用户认证过的页面和菜单选项，而实际上这些仅仅是表示层的访问控制而不能真正生效，攻击者能够很容易伪造请求直接访问未被授权的页面。

编程要求：

- 如果 URL 不是公开的，那么必须限制能够访问的授权用户；
- 加强基于用户或角色的访问控制；
- 完全禁止访问未被授权的页面类型（如配置文件、日志文件、源文件等）；
- 验证你的构架；
- 在每一个层次都使用简单肯定的模型；
- 确保每一层都有一个访问机制；
- 验证你的实现；
- 不能使用自动化的分析工具；
- 确保每个 URL 都被外部过滤器或其他机制保护；
- 确保服务器的配置不允许对非授权页面的访问。

1.4PHP 安全开发要求

1.4.1 变量滥用

PHP-4.1.0 发布的时候建议关闭 register_globals，并提供了 7 个特殊的数组变量来使用各种变量。对于从 GET、POST、COOKIE 等来的变量并不会直接注册成变量，必需通过数组变量来存取。PHP-4.2.0 发布的时候，php.ini 默认配置就是 register_globals=Off。这使得程序使用 PHP 自身初始化的默认值，一般为 0，避免了攻击者控制判断变量。通过以下解决方法实现：

- 配置文件 php.ini 设置 register_globals=Off。
- 要求程序员对作为判断的变量在程序最开始初始化一个值。

1.4.2 文件打开

如非特殊需要，把 php 的文件操作限制在 Web 目录里面。以下是修改 apache 配置文件 httpd.conf 的一个例子：

```
<Directory/usr/local/apache/htdocs>php_admin_
valueopen_basedir/usr/local/apache/htdocs
</Directory>
```

重启 apache 后，/usr/local/apache/htdocs 目录下的 PHP 脚本就只能操作它自己目录下的文件了，否则 PHP 就会报错：

```
Warning:open_basedirrestrictionineffect.Fileisinwrongdirectoryinxxxonlinexx.
```

使用 safe_mode 模式也能避免这种问题，前面已经讨论过。

1.4.3 文件包含

要求程序员包含文件里的参数尽量不能使用变量，如果使用变量，就一定要严格检查要包含的文件名，绝对不能由用户任意指定。如前面文件打开中限 PHP 操作路径是一个必要的选项。另外，如非特殊需要，一定要关闭 PHP 的远程文件打开功能。修改 php.ini 文件：

- allow_url_fopen=Off

- 重启 apache。

1.4.4 文件上传

PHP-4.0.3 以后提供了 is_uploaded_file 和 move_uploaded_file 函数，可以检查操作的文件是否是用户上传的文件，从而避免把系统文件拷贝到 Web 目录。

使用$HTTP_POST_FILES 或$_FILES 数组来读取用户上传的文件变量。严格检查上传变量。比如不允许是 php 脚本文件。把 PHP 脚本操作限制在 Web 目录可以避免程序员使用 copy 函数把系统文件拷贝到 Web 目录。move_uploaded_file 不受 open_basedir 的限制，所以不必修改 php.ini 里 upload_tmp_dir 的值。把 PHP 脚本用 phpencode 进行加密，避免由于 copy 操作泄漏源码。严格配置文件和目录的权限，只允许上传的目录能够让 nobody 用户可写。

对于上传目录去掉 PHP 解释功能，可以通过修改 httpd.conf 实现：

<Directory/usr/local/apache/htdocs/upload>php_flagengineoff

#如果是 php3 换成 php3_engineoff

</Directory>

重启 apache，upload 目录的 php 文件就不能被 apache 解释了，即使上传了 php 文件也没有问题，只能直接显示源码。

1.4.5 命令执行

解决方法：

要求程序员使用 escapeshellcmd()函数过滤用户输入的 shell 命令。启用 safe_mode 可以杜绝很多执行命令的问题，不过要注意 PHP 的版本一定要是最新的，小于 PHP-4.2.2 的都可能绕过 safe_mode 的限制去执行命令。

变量类型缺陷逻辑比较时注意变量类型。

必要的时候使用"==="，这将连变量类型一起比较。

1.4.6 警告及错误信息

修改 php.ini 中关于 Errorhandlingandlogging 部分内容：

error_reporting=E_ALLdisplay_errors=Offlog_errors=Onerror_log=/usr/local/apache/logs/php_error.log

然后重启 apache，注意文件/usr/local/apache/logs/php_error.log，必需可以让 nobody 用户可写。

1.4.7 PHP 与 MySQL 组合的 SQL 注入

解决方法：

要求程序员对所有用户提交的要放到 SQL 语句的变量进行过滤。即使是数字类型的字段，变量也要用单引号扩起来，MySQL 自己会把字串处理成数字。

在 MySQL 里不能给 PHP 程序高级别权限的用户，只允许对自己的库进行操作。

1.4.8 跨站脚本

解决方法：

- 确认输入
- strip_tags()
- htmlspecialchars()
- 清除危险的插入点。

1.4.9 禁用无用的函数

如果觉得有些函数还有威胁，可以设置 php.ini 里的 disable_functions(这个选项不能在 httpd.conf 里设置)，比如：

disable_functions=phpinfo,get_cfg_var

可以指定多个函数，用逗号分开。重启 apache 后，phpinfo,get_cfg_var 函数都被禁止了。建议关闭函数 phpinfo,get_cfg_var，这两个函数容易泄漏服务器信息，而且没有实际用处。

1.4.10 禁用某些类

这个选项是从 PHP-4.3.2 开始才有的，它可以禁用某些类，如果有多个用逗号分隔类名。disable_classes 也不能在 httpd.conf 里设置，只能在 php.ini 配置文件里修改。

1.4.11 限制脚本操作路径

前面分析例程的时候也多次提到用 open_basedir 对脚本操作路径进行限制，这里再介绍一下它的特性。用 open_basedir 指定的限制实际上是前缀，不是目录名。也就是说"open_basedir=/dir/incl"也会允许访问"/dir/include"和"/dir/incls"，如果它们存在的话。如果要将访问限制在仅为指定的目录，用斜线结束路径名。例如："open_basedir=/dir/incl/"。

可以设置多个目录，在 Windows 中，用分号分隔目录。在任何其它系统中用冒号分隔目录。作为 Apache 模块时，父目录中的 open_basedir 路径自动被继承。

1.4.12 其他安全配置

1、取消其它用户对常用、重要系统命令的读写执行权限

一般管理员维护只需一个普通用户和管理用户，除了这两个用户，给其它用户能够执行和访问的东西应该越少越好，所以取消其它用户对常用、重要系统命令的读写执行权限能在程序或者服务出现漏洞的时候给攻击者带来很大的迷惑。记住一定要连读的权限也去掉，否则在 linux 下可以用 /lib/ld-linux.so.2/bin/ls 这种方式来执行。

如果要取消程序如果是在 chroot 环境里，这个工作比较容易实现，否则，这项工作还是有些挑战的。因为取消一些程序的执行权限会导致一些服务运行不正常。PHP 的 mail 函数需要/bin/sh 去调用 sendmail 发信，所以/bin/bash 的执行权限不能去掉。

2、去掉 apache 日志其它用户的读权限：

apache 的 access-log 给一些出现本地包含漏洞的程序提供了方便之门。通过提交包含 PHP 代码的 URL，可以使 access-log 包含 PHP 代码，那么把包含文

件指向 access-log 就可以执行那些 PHP 代码，从而获得本地访问权限；如果有其它虚拟主机，也应该相应去掉该日志文件其它用户的读权限；当然，如果你按照前面介绍的配置 PHP 那么一般已经是无法读取日志文件了。

3、保持运行环境干净。

4、不能在 Web 目录放测试文件。