# Web安全开发规范

产品安全团队

V1.0

|  |  |  |  |
| --- | --- | --- | --- |
| 修订记录 | | | |
| 时间 | 事件 | 修订人 | 审批人 |
| 2016年2月1号 | 新建文档 | 何林如 |  |

目 录

[Web安全开发规范 1](#_Toc445368476)

[第1章 概述 5](#_Toc445368477)

[1.1 目的 5](#_Toc445368478)

[1.2 适用范围 5](#_Toc445368479)

[1.3 适用版本 5](#_Toc445368480)

[1.4 实施 5](#_Toc445368481)

[1.5 例外条款 5](#_Toc445368482)

[第2章 代码质量 6](#_Toc445368483)

[2.1 防范跨站脚本攻击 6](#_Toc445368484)

[2.2 防范SQL注入攻击 10](#_Toc445368485)

[2.3 防止路径遍历攻击 16](#_Toc445368486)

[2.4 防止命令注入攻击 17](#_Toc445368487)

[2.5 防止CSRF漏洞攻击 17](#_Toc445368488)

[2.6 防止其他常见的注入攻击 21](#_Toc445368489)

[2.7 防止URL重定向攻击 21](#_Toc445368490)

[2.8 防止下载敏感资源文件 23](#_Toc445368491)

[2.9 防止上传后门脚本 24](#_Toc445368492)

[第3章 身份与访问控制 24](#_Toc445368493)

[3.1 账户锁定策略 24](#_Toc445368494)

[3.2 登录用图片验证码 25](#_Toc445368495)

[3.3 口令传输 25](#_Toc445368496)

[3.4 保存登录功能 25](#_Toc445368497)

[3.5 纵向访问控制 25](#_Toc445368498)

[3.6 横向访问控制 26](#_Toc445368499)

[3.7 敏感资源的访问 26](#_Toc445368500)

[第4章 会话管理 27](#_Toc445368501)

[4.1 会话超时 27](#_Toc445368502)

[4.2 会话终止 27](#_Toc445368503)

[4.3 会话标识 27](#_Toc445368504)

[4.4 会话标识复用 28](#_Toc445368505)

[第5章 内容管理 28](#_Toc445368506)

[5.1 加密存储敏感信息 28](#_Toc445368507)

[5.2 避免泄露敏感技术细节 28](#_Toc445368508)

[第6章 密码算法 29](#_Toc445368509)

[6.1 安全算法 29](#_Toc445368510)

[6.2 密钥管理 29](#_Toc445368511)

[评审与修订 30](#_Toc445368512)

# 概述

## 目的

本文档规定了平安集团在WEB开发过程中应遵循的安全标准，本文档旨在指导开发人员进行安全开发、并指导安全人员进行Web应用安全检查。

## 适用范围

本配置标准的使用者包括：产品开发团队、安全人员。

本配置标准适用的范围包括：平安集团开发维护管理的Web应用。

## 适用版本

基于B/S架构的Web应用

## 实施

本标准的解释权和修改权属于平安科技银行类移动开发团队产品安全组。

在本标准的执行过程中若有任何疑问或建议，应及时反馈。文档接口人helinru919(何林如)。

本标准发布之日起生效。

## 例外条款

欲申请本标准的例外条款，申请人必须准备书面申请文件，说明业务需求和原因，送交平安科技银行类移动开发团队产品安全组进行审批备案。

# 代码质量

## 防范跨站脚本攻击

|  |  |
| --- | --- |
| 项目名称 | Web应用防范跨站脚本安全 |
| 项目说明 | 系统要防止将用户输入未经检查就直接输出到用户浏览器，防范跨站脚本攻击。  跨站脚本攻击（Cross Site Scripting，缩写为XSS）是指攻击者利用网站应用程序对用户输入验证不足或处置不当，输入内容可以被执行，从而盗取受害者信息，或者使受害者机器执行恶意代码一种攻击方式。  攻击者可以利用存在XSS漏洞的Web网站攻击浏览相关网页的用户，窃取用户会话中诸如用户名和口令（可能包含在Cookie里）等敏感信息、通过插入恶意代码对用户执行挂马攻击、XSS漏洞还可能被攻击者用于网页篡改。跨站脚本漏洞影响所有的Web应用程序框架。 |
| 解决方案 | 在编码过程中使用金丝甲工具、系统自带函数、正则表达式对用户输入、数据参数进行校验和转义，避免外部输入被恶意执行。针对跨站脚本攻击(XSS)，综合使用以下几个措施可以有效防范。  **1、不插入不可信数据**  尽量不往HTML文档中插入不可信数据，除非对不可信数据使用的下文中的编码方法进行了正确的转义或者严密的输入验证。更重要的是，不要接受包含JavaScript脚本的不可信数据并运行该脚本。  <script>...NEVER PUT UNTRUSTED DATA HERE...</script> directly in a script  <!--...NEVER PUT UNTRUSTED DATA HERE...--> inside an HTML comment  <div ...NEVER PUT UNTRUSTED DATA HERE...=test /> in an attribute name  <...NEVER PUT UNTRUSTED DATA HERE... href="/test" /> in a tag name  如果做到这一点，就从源头上避免了跨站脚本攻击，但很多情况，我们需要结合用户输入的数据动态生成HTML文档。  **2、输入验证**  确保在适当位置对用户输入数据进行验证，并检验其属性是否正确，这个措施能阻止恶意用户的输入，使不可信数据变得安全。特别需要注意的是客户端验证容易让攻击者绕过，一定要有服务器端验证。  避免 XSS 漏洞的一个方法是验证所有进入应用程序或由应用程序传送至用户端的数据。由于 Web 应用程序必须验证输入信息以避免其它漏洞（如SQL Injection），因此，可以采取防止注入攻击的解决方案里叙述的类似的验证方法来防止跨站脚本攻击，如白名单验证。严密设计验证规则的白名单验证能够有效保护网站，甚至能够防止未来的新攻击。而由于输入内容的广泛性，黑名单验证方法很难穷举所有危险字符串，是相当脆弱的。建议尽量不要使用黑名单验证的方法。  **3、HTML转义**  如果确实需要往HTML的元素内容中插入不可信数据，确保在插入这些数据前对这些数据进行HTML转义。将要插入的数据进行转义后，能防止改变程序执行上下文，导致恶意代码被执行，如script，style，event handlers等元素中都可包含恶意代码。  采取转义措施后的HTML代码形式如下：  <body>...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...</body>  <div>...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...</div>  转义时，将“<”转换为“&lt;”，下面是常见需要转义的字符：  & --> &amp;  < --> &lt;  > --> &gt;  " --> &quot;  ' --> &#x27;  / --> &#x2F;  仅仅对不可信数据进行HTML转义是不够的。在HTML文档的BODY中，HTML转义能够有效阻止恶意代码的执行。但是在很多地方，如<SCRIPT>标签中、CSS里面、URL、ONMOUSEOVER等事件属性里，HTML转义是起不了作用的。因此，如果网页里所有地方都只进行HTML转义的话，这个网页仍然存在跨站脚本攻击的漏洞。所以必须对HTML的不同部分进行不同的转义。  **4、标签属性转义**  如果确实需要往HTML的通用属性中插入不可信数据，确保在插入这些数据前对这些数据进行属性转义。这些HTML属性值包括width，name，value等。  采取转义措施后的HTML代码形式如下：  <div attr=...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...>content</div> inside UNquoted attribute  <div attr='...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...'>content</div> inside single quoted attribute  <div attr="...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...">content</div> inside double quoted attribute  对于将要插入到HTML通用属性中的数据，将除了字母数字之外的所有ASCII值小于256、采用“&#xHH;”格式的字符进行转义，这样可以阻止插入的数据闭合属性值，从而执行恶意代码。  建议将属性值用引号括起来，这样如果想要中断属性值时，需要相应的引号来闭合，所以能提高插入数据的安全性。如果属性值不使用引号括起来，很多字符能够结束参数值，如[space]，%，\*，+，-，/，;，<，=，>，^，|。  **5、JavaScript转义**  对要插入到HTML中的JavaScript数据，进行JavaScript转义。要关注所有的HTML元素中的JavaScript数据。唯一安全的方式是用引号括起来的数值。任何代码块都是很危险的，所以一定要小心使用转义。  <script>alert('...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...')</script> inside a quoted string  <script>x='...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...'</script> one side of a quoted expression  <div onmouseover="x='...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...'"</div> inside quoted event handler  除了字母和数字，对所有ASCII值小于256、采用“\xHH”格式的字符进行转义，阻止闭合参数值而进入Script脚本的上下文或进入其它属性。不要使用象\”这样的转义快捷键，因为HTML属性解释器可能会匹配这个引号。  建议将事件句柄和属性值用引号括起来，这样如果想要中断事件句柄或属性值时，需要相应的引号来闭合，所以能提高插入数据的安全性。  由于HTML解释器比JavaScript解释器先运行，</script>标签，包括在引号字符串中的</script>将闭合Script语句块。  **6、CSS转义**  当要插入不可信数据到HTML中的样式表或Style标签中时，进行CSS转义。CSS功能强大，被许多攻击利用。所以必须保证不可信数据作为属性的值，而不能进入样式表的其它部分。尽量不要让不可信数据作为复杂的属性，如URL、behavior、custom以及可以包含JavaScript的IE表达式属性等。  <style>selector { property : ...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...; } </style> property value  <span style=property : ...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...;>text</style> property value  除了字母和数字，对所有ASCII值小于256、采用“\HH”格式的字符进行转义，阻止插入的数据闭合属性值，从而执行恶意代码。  由于HTML解释器比JavaScript解释器先运行，</style>标签，包括在引号字符串中的</style>也可能闭合Style语句块。  **7、URL转义**  对要插入到HTML中的URL参数数据，进行URL转义。  <a href="http://www.somesite.com?test=...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...">link</a >  除了字母和数字，对所有ASCII值小于256、采用“%HH”格式的字符进行转义。 |
| 检测步骤 | 检查系统是否存在跨站脚本攻击漏洞。例如在能够回显的输入框输入<script>alert(“xss”)</script> |
| 判断依据 | 要求系统能够将输入内容中的控制字当作纯文本内容处理 |
| 备注 |  |

## 防范SQL注入攻击

|  |  |
| --- | --- |
| 项目名称 | Web应用防范SQL注入安全 |
| 项目说明 | 系统要防止将用户输入未经检查就用于构造数据库查询，防范SQL注入攻击。  SQL注入是注入式攻击的一种，其特征是利用数据库层的漏洞，在输入内容中插入SQL指令，改变SQL语句查询结构，并诱使数据库服务器执行，从而达成攻击目标。  SQL注入属于常见的安全漏洞之一，且通常属于高危安全漏洞，可能造成的危害包括盗取数据库中的敏感信息、修改数据库中的记录、对数据库实施管理性操作（例如关闭数据库系统、执行操作系统命令）等。在实际中，往往会以SQL注入为突破点，结合其他方面的安全漏洞，实施进一步的攻击，从而造成更为严重的危害，例如提升权限控制整个服务器系统、上传木马或其他恶意程序、以服务器为跳板攻击内网其他主机等等。 |
| 解决方案 | 1、使用金丝甲工具或其他安全控件，防御SQL注入攻击。  2、使用参数化查询  参数化查询可预先定义SQL语句查询结构，用户输入的内容则均作为参数传入，这样，数据库即可清晰的区分出SQL指令和用户输入的数据，从而能够阻止对SQL查询结构的篡改，从而有效的防止SQL注入攻击。  在随后的例子中，即使攻击者在输入UserID时键入“1’or ‘1 ‘=’1”，攻击也不会生效，相反，数据库会将“1’or ‘1 ‘=’1”作为要匹配的用户名，试图在库中查找名为“1‘or ‘1 ‘=’ 1”的用户。Java代码如下：  String UserID = request.getParameter("UserID");  String query = "SELECT account\_balance FROM user\_data WHERE UserID = ? ";  PreparedStatement pstmt = connection.prepareStatement( query );  pstmt.setString( 1, UserID);  ResultSet results = pstmt.executeQuery( );  应注意的是，参数化查询仅在正确使用的情况下，才能有效防止SQL注入攻击，一种较常见的错误是，尽管形式上使用了参数化查询，但查询所用的SQL语句系由用户输入内容动态构造而成，当用户输入被嵌入恶意SQL指令时，同样会遭受SQL注入攻击，应加以避免。  使用参数化查询的优点在于简单且有效，而且可以提高数据库操作的效率，对于新开发的应用而言，参数化查询是首选的SQL注入防范措施之一。  参数化查询的缺点在于，无法通过在某一个或某几个点集中部署措施的方式防范SQL注入攻击，相反，必须对程序进行筛查，将所有SQL语句参数化。对于一些已经建成的应用而言，这往往意味着较大的改造工作量，以及较高的改造成本。  3、使用存储过程  在防止SQL注入的功能上，存储过程和参数化查询的作用是一样的，唯一的不同在于存储过程是预先定义并存放在数据库中，被应用程序调用。存储过程中必须保证没有动态生成SQL查询语句。  Java存储过程示例：  String custname = request.getParameter(”customerName”);  try {  CallableStatement cs = connection.prepareCall(”call sp\_getAccountBalance(?)}”);  cs.setString(1，custname);  Result results = cs.executeQuery();  }catch(SQLException se){  //error handling  }  使用存储过程的优点是可以有效防止SQL注入，而且由于存储过程是在数据库中已经编译好的，能提高数据库操作的效率。  存储过程的缺点在于存储过程里的SQL语句也必须使用参数化查询，对于某些数据库系统，使用存储过程的方法没有参数化查询的安全性高。另外，存储过程的保存是在数据库里，不利于软件的移植，也不利于更换数据库。  4、对输入内容进行转义处理  每种数据库系统都有一个或多个转义字符，可以对用户输入的内容进行转义，避免跟数据库系统的元字符冲突。如果对用户的所有输入都进行这样的转义后，数据库系统能够识别用户输入的参数，而不会改变SQL查询语言的结构，所以能够有效防止SQL注入攻击。  可以使用专门的转义组件来对用户输入进行转义，要注意选择经过了严格的安全测试的产品。下面的例子中，以开源组织OWASP提供的开源组件ESAPI来对用户输入进行转义，获取ESAPI的地址如下：  http：//www.owasp.org/index.php/ESAPI  使用ESAPI对用户输入针对ORACLE数据库进行转义的例子如下：  Codec ORACLE\_CODEC = new OracleCodec();  String query = "SELECT user\_id FROM user\_data WHERE user\_name = '" +  ESAPI.encoder().encodeForSQL( ORACLE\_CODEC， req.getParameter("userID")) + "' and user\_password = '"  + ESAPI.encoder().encodeForSQL( ORACLE\_CODEC， req.getParameter("pwd")) +"'";  And it would now be safe from SQL injection， regardless of the input supplied.  For maximum code readability， you could also construct your own OracleEncoder.  Encoder oe = new OracleEncoder();  String query = "SELECT user\_id FROM user\_data WHERE user\_name = '"  + oe.encode( req.getParameter("userID")) + "' and user\_password = '"  + oe.encode( req.getParameter("pwd")) +"'";  对输入内容进行转义处理的优点是可以有效防止SQL注入。  对输入内容进行转义处理的缺点是对于不同的输入内容需要不同的转义，如对Oracle数据库的转义与对SQL Server数据库的转义不一样。转义函数容易出现漏洞，需要选用安全性高的，经过严格验证的转义函数。  5、白名单验证  白名单验证方法是一种值得推荐的方法，可以有效防止注入攻击。该方法首先明确定义什么内容是可以的输入，根据定义的条件执行严格的输入检查，只有满足白名单条件的值才是合法输入，不满足白名单条件的任何值都是禁止接受的输入。如果是结构规整的数据结构，如日期、身份证号、邮政编码、电子邮箱等，开发人员可以定义一个十分强壮的验证模式来检验用户输入，用得最多的是正则表达式。如果用户输入不是结构规整的数据结构，如博客的输入文本等，开发人员也至少可以把用户输入的字符限定于可见字符（或者成为可打印字符）的范围内，确定输入内容的最大长度等。但这种情况会降低安全可靠性。  我们可以采用正则表达式Regex来验证。  using System;  using System.Text.RegularExpressions;  //验证UserAccount是否合法  public bool IsUserAccountValid(string strUserAccount)  {  // 验证是否字符仅仅是有大小写字母及数字构成  // 字符长度在1-20  if ( !Regex.IsMatch(userIDTxt.Text， @"^[a-zA-Z0-9]{1，40}$"))  {  return false;  }else {  return true;  }  }  白名单验证方法的优点是安全系数高，防止SQL注入效果好，甚至能防止未来出现的攻击。事实上，设计上采用白名单方式来验证用户的输入，不仅适用于防范SQL注入攻击，也是安全软件开发中用以防范其它攻击的有效手段。  白名单验证方法的缺点是适用性有限，很多输入不能明确界定合法输入的范围。而且白名单验证方法必须对每个不同类型的输入编写不同的验证函数，不能统一验证。  6、黑名单验证  针对用户输入的参数进行过滤，识别并避免那些列在黑名单中的恶意数据，也可以防止注入攻击。  黑名单验证的问题是：我们需要清楚地知道拒绝哪些危险字符。通常采用的黑名单验证方式中，由于无法穷举所有需要禁止的字符，难免存在一些漏洞；另外，攻击者往往可以通过“改头换面”的办法来绕过黑名单，常见的手段是采用另一种编码方式，如采用%0x来编码等等。所以黑名单验证并不能保证完全防止SQL注入攻击。  采用黑名单验证要注意避免这些问题。如果对于特定环境下，恶意数据可以明确界定，也不能通过转义来绕过的话，黑名单是可以防止注入攻击的。  例如，SQL语法中有下面这些字符需要特殊注意：  SQL语句中的含义  ; 查询分隔符  ' 数据分隔符.  -- 注释分隔符.  /\* ... \*/ 注释分隔符.  AND 逻辑与  OR 逻辑或  NOT 逻辑非  xp\_ 指示扩展的stored procedures，例如 xp\_cmdshell.  ... ...  所以在验证用户输入的开发过程中，推荐采用白名单的设计方式来确认用户的输入是否是有效的。  下面给出Java过滤程序的防注入函数原型：  public static boolean sql\_inj(String str)  {  String inj\_str = "'|and|exec|insert|select|delete|update|count|\*|%|chr|mid|master|truncate|char|declare|;|or|-|+|,";  String inj\_stra[] = split(inj\_str, "|");  for (int i=0 ; i < inj\_stra.length ; i++ )  {  if (str.indexOf(inj\_stra[i])>=0)  {  return true;  }  }  return false;  }  Java中调用该函数检查是否包函非法字符  if(request.getParameter("userID") != null)  userID = request.getParameter("userID").trim();  if (StringUtil.sql\_inj(userID) || StringUtil.sql\_inj(pwd)){  System.out.println ('参数中包含非法字符！');  }else{  ……  }  黑名单验证方法的优点是适用性非常广泛，而且黑名单验证方法可以对不同类型的输入应用相同的验证函数，改进已有软件系统时添加黑名单验证方法也比较方便。  黑名单验证方法的缺点是防止SQL注入不彻底，容易漏掉需要过滤的字符串，从而出现漏洞；黑名单中禁止的危险字符串列表可能很长，会降低程序的效率。  7、其它  数据库访问帐号的最小化权限设计原则，可以有效降低SQL注入攻击的危害性。  最小化权限的设计，同白名单一样，也是安全软件开发中的常用的设计原则。应用到SQL注入攻击环境下就是：应用程序用来访问数据库的帐号，所拥有的权限应该最小化。  需要强调的一点是，SQL注入攻击的根源在于Web开发中的不正确的代码。最小化权限的设计并不能替代修复代码本身错误的工作。它只是作为深层防御（Defense In Depth）中的缓解手段，其目的是为了尽可能降低SQL注入攻击的危害性。  例如，如果是通过Windows authentication机制来连接并访问数据库的话，用来访问数据库的Windows的帐号的权限就应该受到限制。一个非管理员权限的帐号，就可以阻止执行Fdisk类似级别的操作。 |
| 检测步骤 | 检查系统是否存在SQL注入漏洞。例如在输入框中输入’ |
| 判断依据 | 系统要使用诸如prepared statement等方式防止SQL注入，将输入内容中的控制字也当作纯文本处理 |
| 备注 |  |

## 防止路径遍历攻击

|  |  |
| --- | --- |
| 项目名称 | Web应用防范路径遍历安全 |
| 项目说明 | 系统要防止将用户输入未经检查就用于构造文件路径，防止路径遍历攻击。 |
| 解决方案 | 1、设定禁止使用目录索引文件。某些Web服务器在接收到用户对一个目录的访问时，会查找指定的目录索引文件。默认情况下该文件是index.html。如果该文件不存在，那么Web服务器会创建动态列表为用户显示该目录的内容。通常这样的设置会暴露Web站点结构，因此需要修改配置文件禁止显示动态目录索引。  2、禁止默认访问。一个好的安全策略是禁止默认访问的存在。首先禁止默认访问，只对指定目录开启访问权限。  3、禁止用户重载。为了禁止用户对目录配置文件（.htaccess）进行重载（修改）。 |
| 检测步骤 | 尝试在URL与输入中构造文件路径并查看页面反应 |
| 判断依据 | 不允许通过构造文件路径的方式直接查看文件 |
| 备注 |  |

## 防止命令注入攻击

|  |  |
| --- | --- |
| 项目名称 | Web应用防范命令注入安全 |
| 项目说明 | 系统要防止将用户输入未经检查就用于构造操作系统命令并执行。 |
| 解决方案 | 1、 在程序中禁止用户直接控制命令的执行，采用权限最小原则控制执行，并通过白名单的方式允许用户输入字符和命令。  2、 利用 可依赖的变量在执行赛程中构建绝对路径，防止攻击者通过修改程序运行命令的环境来控制命令的执行。  3、 对从配置文件和环境变量中读取出来的命令变量和路径进行校验，检查期合法性、用户及权限，避免资源被恶意篡改的情况下执行。  4、在代码级调用shell的时候对命令行特殊字符进行转义（|、&、；等），防止执行其他非法命令 |
| 检测步骤 | 尝试在各个输入点进行命令注入攻击 |
| 判断依据 | 命令注入攻击不得成功 |
| 备注 |  |

## 防止CSRF漏洞攻击

|  |  |
| --- | --- |
| 项目名称 | Web应用防范CSRF安全 |
| 项目说明 | 跨站请求伪造（CSRF）的英文全称是Cross Site Request Forgery，字面上的意思是跨站点请求伪造。 CSRF是诱骗受害者在不知情的情况下向一个易受攻击的Web应用程序发送请求，达到攻击者目标的一种攻击行为。  大多数情况，浏览器会自动将认证信息跟请求的网站联系起来，如用户的会话Cookie，认证凭证，IP地址等。因此，如果用户已经通过网站的认证的话，站点没有办法识别哪个请求是合法的。  一个利用跨站请求伪造漏洞的攻击者会强迫一个已经被他控制的用户浏览器，向要攻击的目标Web站点发送特殊的请求，依靠Web站点对受害用户的信任，它就允许攻击者以受害用户的身份执行相应的动作。例如创建账号、购买商品、转帐和下载资料等。这是由于Web站点只根据用记帐号或密码的方式进行确认，如COOKIE方式，而这种方式很容易被伪造，Web站点还应当在用户登录后进行进一步的操作权限确认。  有时CSRF攻击会保存到有漏洞网站自身上，这种漏洞属于站内型CSRF漏洞。这种漏洞的实现可能是存储了攻击者的HTML标签，如IMG、IFRAME等；或者通过更复杂的CSS攻击实现。如果CSRF攻击存储在一个网站上，攻击的危险性将被放大，因为受害者正在访问该网站，进而访问到该网站中含有CSRF攻击的网页的可能性很大。  这个漏洞在所有Web应用程序框架中都存在。这些攻击被誉为基于Web漏洞中的“沉睡的巨人”，因为互联网上的许多站点对此毫无防备，同时还因为这类攻击一直为Web开发和安全社区所忽视。 |
| 解决方案 | **1、一次性令牌**  使用会话 Cookie 的应用程序必须在每个表单发布中包含某些信息，以便后台代码可以用来验证请求的来源。其中一种方法是使用一次性令牌，使得在任何情况下令牌都不能够被猜测或者伪装，一次性令牌在完成它的工作后将被销毁。  开发者可根据当前的时间（毫秒数）获得一个唯一的ID，这个唯一ID有利于生成一个不重复的数值。  <form action="/transfer.do" method="post">  <input type="hidden" name="CSRFToken" value="OWY4NmQwODE4ODRjN2Q2NTlhMmZlYWEwYzU1YWQwMTVhM2JmNGYxYjJiMGI4MjJjZDE1ZDZjMTViMGYwMGEwOA==">  …  </form>  这样，后端逻辑可先验证请求标识符，然后再处理其他表单数据。每个新表单中的请求标识符都可以是唯一的，也可在特定会话的各个表单之间共享。对于会话标识符而言，攻击者越难猜出请求标识符，则越难成功进行 CSRF 攻击。  开发人员要注意所有可能执行 POST 操作的 HTML 表单和 XMLHttpRequest 对象。必须确定每个表单是否会让攻击者定为 CSRF 目标，以及是否采取了适当的缓解技术。  一次性令牌方法的优点是安全性高，能有效防止CSRF攻击；用户体检好，不需要用户做额外工作。  一次性令牌方法的缺点是实现过程比较困难，特别是对于已上线的软件系统，不方便修改。  **2、添加验证码**  另外一个解决这类问题的方法则是在用户提交的每一个表单中使用一个随机验证码，让用户在文本框中填写图片上的随机字符串，并且在提交表单后对其进行检测。这个方法曾经存在风险，这是由于验证码图片的使用涉及了一个被称为MHTML的Bug，但这个Bug已经被微软放出的Windows XP和Windows Vista及其浏览器IE6.0的修复包所解决。  添加验证码方法的优点是实现简单，也能有效防止CSRF。  添加验证码方法的缺点是用户体验变差，用户需要增加输入验证码的步骤。  **3、提交两个Cookie**  提交两个Cookie的方法就是在每次提交FORM请求时，用两种不同的方式提交Cookie。第一种方式作为普通的请求头，第二种方式作为一个隐藏的FORM字段值。当用户访问一个网站时，这个网站通常会生成一个强解密的伪随机数，作为Cookie返回给用户端，这个伪随机数通常会作为会话ID。当一个POST请求提交到服务器，只有在Cookie值和FORM中的隐藏字段值相等的时候才认为请求时有效的。当攻击者伪装成受害者提交请求时，攻击者只能修改FORM中的值，不能读取和修改Cookie中的值，所以他不能将FORM中的值设成跟Cookie相等的值，除非攻击者能够猜出会话ID的值。  事实上，如果不考虑用户的Cookie容易由于网站中存在XSS漏洞而被偷窃这一情况，这是一个很好的应对对CSRF的解决方案。如果我们为用户的每一个表单请求中都加入随机的Cookie，那么这种方法会变得更加安全。  提交两个Cookie的方法的优点是实现比较简单；用户体检好，不需要用户做额外工作。  提交两个Cookie的方法的缺点是在Cookie容易被偷窃的情况下不能有效防止CSRF攻击，增加了会话劫持的风险。 |
| 检测步骤 | 以下是一个跨站请求伪造漏洞的示例。该示例中，网站A的一个 Web 应用程序允许管理员通过提交此表单来创建新账户，代码如下所示：  <form method="POST" action="/new\_user" >  Name of new user：<input type="text" name="username">  Password for new user：<input type="password" name="user\_passwd">  <input type="submit" name="action" value="Create User">  </form>  攻击者可能会使用以下内容来建立一个网站B：  <form method="POST" action="http://www.example.com/new\_user">  <input type="hidden" name="username" value="hacker">  <input type="hidden" name="user\_passwd" value="hacked">  </form>  <script>  document.usr\_form.submit();  </script>  如果 example.com 网站的管理员在网站A上具有活动会话时访问了恶意网站B的页面，则他会在毫不知情的情况下为攻击者创建一个账户，因为此时管理员还保持着网站A的会话，恶意网站B向网站A提交的请求会被接受，这就是 CSRF 攻击过程。正是由于该应用程序无法确定请求的来源，任何请求都有可能是用户选定的合法操作，也有可能是攻击者设置的恶意操作，这样就有可能受到 CSRF 攻击。攻击者无法查看伪请求生成的网页，因此，这种攻击技术仅适用于篡改应用程序状态的请求。  大多数 Web 浏览器会随每次请求发送一个名为 referer 的 HTTP 头文件。referer 头文件应该包含参考页面的 URL，但攻击者可以伪造，因此无法利用 referer 头文件确定请求的来源。  如果应用程序通过 URL 传递会话标识符（而不是 Cookie），则不会出现 CSRF 问题，因为攻击者无法访问会话标识符，也无法在伪请求中包含会话标识符。 |
| 判断依据 | 无法执行恶意操作。 |
| 备注 |  |

## 防止其他常见的注入攻击

|  |  |
| --- | --- |
| 项目名称 | Web应用防范其它注入安全 |
| 项目说明 | 防止系统存在LDAP注入、XML注入、XPATH注入、SMTP注入等漏洞。 |
| 解决方案 | 对用户在输入进行校验和转义，避免被恶意执行。 |
| 检测步骤 | 尝试在各个输入点进行其它常见注入攻击 |
| 判断依据 | 各类注入攻击不得成功 |
| 备注 |  |

## 防止URL重定向攻击

|  |  |
| --- | --- |
| 项目名称 | Web应用防范URL重定向安全 |
| 项目说明 | 在Web应用中重定向是极为普遍的，并且通常重定向所引向的目的是用户输入的目标URL，而如果这些重定向目标未被验证，那么攻击者就可以引导用户访问攻击者建立的恶意站点，这就存在重定向漏洞。  同样，转发也是极为普遍的，本质上转发是在同一个应用中对一个新页面发送请求，并且有时是用参数来定义目标页面的。同样，如果参数未被验证，那么攻击者就可以利用其来绕过认证或是授权检查。  最终造成的后果，重定向会使得用户访问钓鱼网站或是其它类型恶意网站，而转发则会让攻击者利用先前安全检查过的请求来绕过认证或授权。 |
| 解决方案 | 1、避免使用重定向和转发机制  不使用重定向和转发机制，当然能够避免这种机制带来的漏洞。而一般情况下，都可以找到替代办法。开发人员可以从应用程序中删除重定向页面，用直接指向相关目标URL的链接替代指向重定向页面的链接。  避免使用重定向和转发机制的优点是能有效避免重定向攻击。  避免使用重定向和转发机制缺点是适用范围有限，有些应用需要使用重定向机制。  2、目标URL不包含用户参数  如果确实需要使用重定向机制，尽量在服务器端提供映射机制，给用户端只提供一个选择目标URL的索引，用户选择了目标URL参数后，在服务器端将参数转变为真正的目标页面。在这个过程中，有效隔离了用户输入的参数和重定向的目标，没有将用户输入的参数合并到重定向的目标URL中，所以也能有效防止重定向攻击。  采用目标URL不包含用户参数的方法的优点是适用范围比较广，而且由于这种方法有效隔离了用户输入的参数和重定向的目标，所以安全性也很高，能有效防止重定向攻击。  采用目标URL不包含用户参数的适用范围有限，有些应用需要用户输入目标URL，而不是事先能确定的URL。  3、验证用户参数  如果重定向页面不可避免地要收到用户提交的输入并将它合并到重定向目标中，应使用以下措施降低重定向攻击的风险。  1) 应用程序应在所有重定向中使用相对URL，重定向页面应严格确认它收到的URL是一个相对URL。应当确认：用户提交的URL或者以单独一个斜线字符、后接一个字母开头，或者以一个字母开头，并且在第一个斜线前没有冒号。应拒绝而不是净化任何其他输入。  2) 重定向页面在发布重定向之前，应确认提交的URL以Http://yourdomainname.com/ 开头。否则应用程序应该在所有重定向中使用相对于Web根目录的URL，在发布重定向之前，重定向页面在所有用户提交的URL前附加http://yourdomainname.com。或者http://yourdomainname.com/ （如果用户提交的URL并不以斜线字符开头）。  3) 与基于DOM的XSS漏洞一样，建议应用程序不要根据DOM数据通过客户端脚本执行重定向，因为这些数据不在服务器的直接控制范围内。  采用验证用户参数方法的优点是适用范围广泛，几乎能满足所有应用重定向的场景。  采用验证用户参数方法的缺点是处理方法复杂多样，而且容易出现缺陷，从而不能完全防止重定向攻击，因此安全性较差。 |
| 检测步骤 | 本示例中，服务器端程序根据用户输入的Url决定转跳目的地，Jsp代码如下  <form method="GET" action="/user" >  Destination Url：<input type="text" name="target">  </form>  正常用户在文本框中输入要转跳的URL，如http://www.example.com/index.jsp，这时的请求为：  http://www.example.com/redir.jsp?target=http://www.example.com/index.jsp  但是，如果攻击者直接将URL修改为：  http://www.example.com/redir.jsp?target=http://www.malicious.com/index.jsp  用户看到这个链接指向了一个可信网站，没有注意到重定向机制可能将用户转到恶意网站，攻击者可能成功诱使用户点击该请求，则将该用户带到攻击者的恶意网站了。而且攻击者还可以将上面的危险链接进行编码：  http://www.example.com/redir.jsp?target=http://%77%77%77%2e%6d%61%6c%69%63%69%6f%75%73%2e%63%6f%6d%2f%69%6e%64%65%78%2e%6a%73%70  这时不能直接看到重定向的目的URL，从而使这个危险链接更具隐蔽性。 |
| 判断依据 | 无法修改跳转URL |
| 备注 |  |

## 防止下载敏感资源文件

|  |  |
| --- | --- |
| 项目名称 | Web应用防范下载漏洞安全 |
| 项目说明 | 如果系统提供了下载功能，要防止用户通过路径遍历漏洞下载敏感资源文件。 |
| 解决方案 | 对敏感资源进行权限控制和加密，避免被恶意获取。 |
| 检测步骤 | 如果系统提供了下载功能，试图通过路径遍历漏洞下载敏感资源文件。 |
| 判断依据 | 各类下载攻击不得成功 |
| 备注 |  |

## 防止上传后门脚本

|  |  |
| --- | --- |
| 项目名称 | Web应用防范上传漏洞安全 |
| 项目说明 | 如果系统提供了文件上传功能，要防止用户上传后门脚本。 |
| 解决方案 | 文件上传应做到：  1、 上传文件类型和格式校验。  2、 上传文件以二进制形式下载，不提供直接访问。  3、 对上传的文件及文件夹进行读、写、执行权限限制，保证权限最小化。 |
| 检测步骤 | 如果系统提供了上传功能，试图通过上传功能上传恶意文件。 |
| 判断依据 | 各类上传攻击不得成功 |
| 备注 |  |

# 身份与访问控制

## 账户锁定策略

|  |  |
| --- | --- |
| 项目名称 | Web应用账户锁定策略安全 |
| 项目说明 | 用户登录失败一定次数后系统自动锁定账号一段时间，以防止暴力猜测密码。 |
| 检测步骤 | 尝试使用错误用户名口令失败登录多次， |
| 判断依据 | 用户登录失败一定次数后系统自动锁定账号。 |
| 备注 |  |

## 登录用图片验证码

|  |  |
| --- | --- |
| 项目名称 | Web应用登录验证策略安全 |
| 项目说明 | 用户登录需提供图片验证码，以防止固定密码暴力猜测账号。 |
| 检测步骤 | 检查登录认证界面输入项，并右键点击图片查看链接属性。 |
| 判断依据 | 要求包含图片验证码输入项，并且图片链接属性不得包含明文图片验证码。 |
| 备注 |  |

## 口令传输

|  |  |
| --- | --- |
| 项目名称 | Web应用口令传输策略安全 |
| 项目说明 | 不能明文传输用户登录密码。 |
| 检测步骤 | 尝试登录系统，并使用抓包工具查看交互过程中在网络传输的内容。 |
| 判断依据 | 要求不得出现明文口令 |
| 备注 |  |

## 保存登录功能

|  |  |
| --- | --- |
| 项目名称 | Web应用保存登录安全 |
| 项目说明 | 不能提供“保存登录”功能，该功能可能被利用于CSRF攻击。 |
| 检测步骤 | 检查登录界面是否提供了保存登录功能 |
| 判断依据 | 不得提供该功能。 |
| 备注 |  |

## 纵向访问控制

|  |  |
| --- | --- |
| 项目名称 | Web应用纵向访问安全 |
| 项目说明 | 合理进行纵向访问控制，不允许普通用户访问管理功能。 |
| 检测步骤 | 了解是否有不允许普通用户访问的功能，尝试直接在浏览器中访问功能链接。 |
| 判断依据 | 用户不得跨权限访问受控页面 |
| 备注 |  |

## 横向访问控制

|  |  |
| --- | --- |
| 项目名称 | Web应用横向访问安全 |
| 项目说明 | 合理进行横向访问控制，不允许用户访问其他用户的敏感数据。 |
| 检测步骤 | 了解是否存在敏感信息，检查是否对个人敏感信息进行了有效保护 |
| 判断依据 | 用户不得跨权限查看其它用户受保护敏感信息 |
| 备注 |  |

## 敏感资源的访问

|  |  |
| --- | --- |
| 项目名称 | Web应用敏感资源访问安全 |
| 项目说明 | 需要限制对敏感资源的访问，例如后台管理，日志记录等。 |
| 检测步骤 | 检查服务器的文件是否存在敏感资源，测试是否限制了这些资源的访问。 |
| 判断依据 | 对敏感资源的访问应当受控。 |
| 备注 |  |

# 会话管理

## 会话超时

|  |  |
| --- | --- |
| 项目名称 | Web应用会话超时安全 |
| 项目说明 | 当用户长时间不操作时，系统自动终止超时会话。 |
| 检测步骤 | 登录系统后不操作，等待合理的时间间隔。 |
| 判断依据 | 要求预先设计的时间间隔后查看页面自动中止超时会话。 |
| 备注 |  |

## 会话终止

|  |  |
| --- | --- |
| 项目名称 | Web应用会话终止安全 |
| 项目说明 | 系统需提供“退出”功能，允许用户强制终止当前的会话。 |
| 检测步骤 | 登录系统后点击系统提供的“退出”功能，然后在同一IE窗口下视图回退到登录后的页面，并访问相应的功能 |
| 判断依据 | 点击退出后，上述检测操作结果不成功 |
| 备注 |  |

## 会话标识

|  |  |
| --- | --- |
| 项目名称 | Web应用会话标识安全 |
| 项目说明 | 会话标识必须足够随机，防止攻击者猜测标识或依据当前标识推导后续的标识。 |
| 检测步骤 | 检查多个会话标识的格式。 |
| 判断依据 | 多个会话标识不得存在简单明了的逻辑关系，要求具有随机性 |
| 备注 |  |

## 会话标识复用

|  |  |
| --- | --- |
| 项目名称 | Web应用会话标识复用安全 |
| 项目说明 | 用户登录后必须分配新的会话标识，不能继续使用用户未登录前所使用的标识。 |
| 检测步骤 | 检查登录前后是否使用相同的会话标识。 |
| 判断依据 | 用户登录后必须分配新的会话标识，不能继续使用用户未登录前所使用的标识。 |
| 备注 |  |

# 内容管理

## 加密存储敏感信息

|  |  |
| --- | --- |
| 项目名称 | Web应用加密存储敏感信息 |
| 项目说明 | 系统应当加密存储敏感信息，如密码、信用卡号等。 |
| 检测步骤 | 分析系统中敏感信息的存储与加密 |
| 判断依据 | 要求加密算法安全，对信息有适当访问控制 |
| 备注 |  |

## 避免泄露敏感技术细节

|  |  |
| --- | --- |
| 项目名称 | Web应用信息泄漏 |
| 项目说明 | 系统应当避免向用户提示过多的技术细节，防止被攻击者利用。例如错误信息中可能包含SQL语句，这有利于攻击者构造合法的攻击字串；又如Html中可能包含了技术性的注释语句，可能被攻击者利用。 |
| 检测步骤 | 分析各个页面的源码，查看提示页面，尤其是出错提示页面 |
| 判断依据 | 各个页面不得包含技术性注释，各个提示页面不得包含Web服务器版本、源代码等信息 |
| 备注 |  |

# 密码算法

## 安全算法

|  |  |
| --- | --- |
| 项目名称 | Web应用安全算法 |
| 项目说明 | 如果系统采用了密码算法，应当采用安全的密码算法，且符合算法的应用场景。 |
| 检测步骤 | 检查所有系统中使用的安全算法 |
| 判断依据 | 不得使用已经被证明为不安全的算法或者自定义不安全算法 |
| 备注 |  |

## 密钥管理

|  |  |
| --- | --- |
| 项目名称 | Web应用密钥管理 |
| 项目说明 | 如果系统采用了密码算法，且拥有密钥，那么应当有文档化的密钥管理办法并严格遵照执行。 |
| 检测步骤 | 检查所有系统中使用的密钥管理 |
| 判断依据 | 不得使用不安全的密钥管理办法 |
| 备注 |  |

# 评审与修订

本标准由平安科技银行类移动开发团队产品安全组定期进行审查，根据审视结果修订标准，并颁发执行。