WeLink 开放平台安全开发规范 V1.0

华为技术有限公司

Huawei Technologies Co., Ltd.

目　录

版本	拟制/修订责任人	拟制/修订日期	修订内容及理由
V01.00	WeLink安全团队	2019-08-15	新建

1 概述

1.1 背景简介

在 Internet 飞速演变的今天， 网络安全所面临的挑战日益严峻，其中尤其是 Web 安全应用。许多设计和开发人员不清楚如何开发安全的应用程序，常常由于设计和编码不当，开发出来的应用存在较多的安全漏洞，这些安全漏洞一旦被黑客利用将导致严重甚至是灾难性的后果。为了帮助应用开发人员开发安全的 WeLink 应用（ISV、小程序、We 码），减少甚至规避由于设计和编码不当引入的安全风险，提高应用安全性，特制定本规范。

下表列出了一些 Web 缺陷类别，并针对每类缺陷列出了由于设计不当可能会导致的潜在问题。针对这些潜在的问题，本规范中有相应的解决措施。

表1 应用程序缺陷和由于不良设计可能导致的问题

缺陷类别	由于不良设计可能导致的问题
身份验证	身份伪造、口令破解、权限提升和未授权访问。
权限管理	访问机密或受限数据、篡改和执行未授权操作。
输入检验	通过嵌入查询字符串、窗体字段、Cookie 和 HTTP 标头中的恶意字符串来执行攻击。包括命令执行、跨站点脚本编写 (XSS)、SQL 注入和缓冲区溢出攻击等。
参数操作	路径遍历攻击、命令执行、此外还有旁路访问控制机制、导致信息泄露、权限提升和拒绝服务。
敏感数据	机密信息泄漏和数据篡改。
加密技术	未授权访问机密数据或帐号信息。

1.2 适用范围

本规范的制定考虑了各种应用开发的共性，适合于 WeLink 开放平台接入的应用。要求 WeLink 开放平台的应用遵循。除非特别说明，本规范中所有的示例代码均为 Java 语

言。本规范的读者及使用对象主要为 WeLink 开放平台的需求分析人员、设计人员、开发人员、测试人员等。

1.3 术语定义

- ✧ **规则：**接入 WeLink 开放平台的应用需要遵守的约定
- ✧ **说明：**对此规则或建议进行相应的解释

2 认证

规则2.1 对用户的最终认证处理过程必须在服务器端进行。

说明：客户端认证很容易被绕过，禁止仅仅通过脚本或其他形式在客户端进行验证，必须在应用服务器进行最终认证处理（如果采用集中认证，那么对用户的最终认证就是放在集中认证服务器进行）。

规则2.2 认证失败后，不能提示给用户详细以及明确的错误原因，只能给出一般性的提示，同时记录日志信息。

说明：认证错误不应给出明确的提示，否则可被攻击者利用来枚举用户名和口令。可以提示：“登录失败；无效的用户名或口令，”；不能提示：“用户foo：口令无效”、“登录失败，无效用户ID”、“登录失败；帐号已停用”、“登录失败；用户未激活”等。

规则2.3 禁止在系统中预留任何未公开帐号或特殊的访问机制。

说明：产品若存在预留的未公开帐号或特殊的访问机制，有可能被内部恶意人员或外部人员获知进行私自操作，存在绕过鉴权进入系统的风险。

规则2.4 对于重要的管理事务或重要的交易事务要进行重新认证或二次认证。

说明：重要的管理事务，比如重新启动业务模块；重要的交易事务，比如转账、余额转移、充值等。重新认证，比如让用户重新输入口令；二次认证，比如让用户输入动态密码/短信随机密码。重新认证或二次认证，可以减小会话劫持给用户带来损失，还可以减轻跨站请求伪造攻击。

3 授权

规则3.1 对于每一个需要授权访问的请求都核实用户的会话标识是否合法、用户是否已经

被明确授权执行该操作。

说明：使用 Web 框架内置的授权检查、或者采用自己实现集中的授权模块来进行授权操作，如果用户会话标识（会话标识使用 Session identifiers，Session ID 有其他的含义）不合法，或者用户没有被授权执行该操作，则拒绝请求，并记录日志。防止用户直接在地址栏中输入 URL，越权访问某些页面或 servlet。

规则3.2 默认对于每一个请求都进行鉴权，避免 URL 越权，对不需要鉴权的采用白名单的方式；对于用户请求的 URL 匹配规则设计要严谨，避免鉴权被绕过。

说明：客户所有的请求，包括私有资源请求（图片，JS，文件）默认都需要鉴权，对不需要鉴权的请求可以设置白名单，白名单中的 URL 匹配尽量使用使用完全匹配，保证 URL 鉴权不被绕过。不能使用简单的包含某个字符串的匹配算法，这样的算法容易被找出漏洞而导致鉴权被绕过，建议使用完全匹配的方式进行鉴权，对没有匹配上的请求，又不在白名单列表中的，拒绝访问，如果鉴权 URL 有共性，建议使用通配符匹配。

规则3.3 对于用户请求的 URL 需要先标准化后进行鉴权，避免鉴权被绕过

说明：客户的请求 URL 是可以任意输入的，通常攻击者利用 URL 编码、”../”,”./”,”//”路径、”;”作为结尾符构造等方式伪造异常 URL 请求，从而绕过鉴权，导致系统不受保护。

标准化(参考：https://en.wikipedia.org/wiki/URL_normalization)：去除“..” 、 “.”、”//”、默认页面、大小写一致性、去空字符串、URL 解码、等返回用户访问的真实资源。

备注：禁止直接调用函数 HttpServletRequest.getRequestURI()，HttpServletRequest.getRequestURL()获取 URL 进行鉴权，这两方法获取 URL、URI 未做标准化，可能被用于绕过鉴权。

规则3.4 对于受保护资源的访问，必须检查用户是否拥有访问该资源的权限，如果没有则拒绝访问并记录日志。

说明：许多应用程序会检查用户是否有权使用一个特定的功能，但是没有检查用户是否被允许访问所请求的资源。例如，论坛可以检查用户是否允许回复以前的消息，但没有检查所回复的消息是在一个受保护的或隐藏的论坛或主题中；或网上银行应用程序可能会检查用户是否有权汇款，但是没有验证“转出账号”是否属于该用户。

规则3.5 采用基于角色的访问控制机制，授权和用户角色数据必须存放在服务器端，不能存放在客户端，鉴权处理也必须在服务器端完成。

说明：禁止将授权和角色数据存放在客户端中（比如 cookie 或隐藏域中），防止被篡改。

规则3.6 对于运行应用程序的操作系统帐号、数据库帐号和中间件帐号，在满足业务需求的前提下，必须使用最低级别权限的账号，不应使用“root”、“admin”等特权帐号或高级别权限帐号，应该尽可能地使用低级别权限的帐号。

说明：根据业务系统要求，创建相应的帐号，并授予必需的权限。不能使用“root”、“admin”等管理帐号或高级别权限帐号。

规则3.7 当授权变更、终止或失效时（例如，权限、角色或业务流程等发生变化），应用程序应该支持禁用帐号和终止会话的能力。

说明：当用户离职应该及时终止其对应的会话，并禁用或删除该用户；当用户的权限、角色或业务流程发生变更时，应该能够终止用户对应的会话，以及时应用新权限、新角色或新流程。

4 会话管理

规则4.1 会话过程中不允许修改的数据信息，必须作为会话状态一部分在服务器端存储和维护。

说明：会话过程中不允许修改的信息，例如，当用户通过认证后，其用户标识在整个会话过程中不能被篡改；又如，用户网上购物时的商品价格也是不允许被篡改。这些信息是禁止通过隐藏域或 URL 重写等不安全的方式存储和维护。对 Java 语言，可以通过 session 对象进行存储和维护，或者直接查询数据库获取。

规则4.2 所有登录后才能访问的页面都必须有明显的“注销（或退出）”的按钮或菜单，如果该按钮或菜单被点击，则必须使对应的会话立即失效。

说明：这样做是为了让用户能够方便地、安全地注销或退出，减小会话劫持的风险，减少服务器端内存占用。

规则4.3 当 Web 应用跟踪到非法会话，则必须清除会话并返回到认证界面。

说明： 非法会话的概念就是通过一系列的服务端合法性检测（包括访问未授权资源，缺少必要参数等情况），最终发现的不是正常请求产生的会话。

5 输入校验

输入校验指的是一组验证确保所有输入数据符合应用程序所预期的类型、长度、范围、允许的字符集，以及不包含已知危险字符的处理过程。

很多常见的web应用安全漏洞都是因为未对来自客户端或者环境的输入做有效的校验所导致，例如各种类型的解释器注入、文件系统攻击以及缓冲区溢出等。为了保证系统的健壮性，永远不要信任那些来自客户端、第三方系统等外部实体的数据，在使用这些数据之前须进行输入校验以确保其是可用的且不会对系统造成危害。

规则5.1 对所有来自不可信数据源的数据进行校验，拒绝任何没有通过校验的数据。

说明：非系统本身直接可控组件生成的数据是不可信的，如来自客户端、第三方系统、外部接口的数据。这些数据可能是恶意的，如果不进行校验，可能导致注入、XSS等多种攻击。来自客户端的数据包括text、password、textareas、file表单域、hidden fields、selection boxes、check boxes、radio buttons、cookies、HTTP headers、localStorage、sessionStorage、Web SQL database、Indexed database、热点链接包含的URL参数的数据等。

规则5.2 不能依赖于客户端校验，必须使用服务端代码对输入数据进行最终校验。

说明：客户端的输入限制或者校验可能会被攻击者绕过。例如，攻击者可以通过禁用浏览器的JavaScript来使客户端代码失效，或者直接采用代理软件拦截并修改输入参数名称和值。客户端的校验只能作为辅助手段，可以减少客户端和服务端的信息交互次数，提高响应速度。

规则5.3 校验输入数据的类型、格式、长度和范围。

说明1：这里的类型包括基本的数据类型（整型、浮点型、字符型等），同时也包括复杂的构造数据类型，例如，email地址、ip地址、日期或者是自定义的数据格式等。

说明2：必须校验输入数据的长度是否符合要求，长度校验会加大攻击者实施攻击的难度。

说明3：这里的范围可以是数值范围或者是集合范围。如果输入数据是数值，必须校验数值的范围是否正确，如年龄应该为0～150之间的正整数；如果输入数据是在某个特定集合范围内，必须校验实际输入是否在集合内，如性别应该是男、女。

规则5.4 禁止采用未经校验的输入参数直接串联来构造可执行 SQL 语句。

说明：例如，string sql = "select status from Users where UserName='" + txtUserName.Text +

";这样很容易被SQL注入攻击。对于java/JSP语言，使用预编译语句PreparedStatement代替直接的语句执行Statement。使用预编译语句PreparedStatement，类型化 SQL 参数将检查输入的类型，确保输入值在数据库中当作字符串、数字、日期或boolean等值而不是可执行代码进行处理，从而防止SQL注入攻击。而且，由于 PreparedStatement 对象已预编译过，所以其执行速度要快于 Statement 对象。因此，多次执行的 SQL 语句经常创建为 PreparedStatement 对象，还可以提高效率。

参考如下代码：

```
String inssql = "insert into buy(empid, name, age, birthday) values (?,?,?,?)";
PreparedStatement stmt = null;
stmt = conn.prepareStatement(inssql);
stmt.setString(1, empid);
stmt.setString(2, name);
stmt.setInt(3, age);
stmt.setDate(4, birthday);
stmt.execute();
```

备注：使用 like 进行模糊查询时，如果直接用"select * from table where comment like %?%"，程序会报错，必须采用如下方法：

```
String express = "select * from table where comment like ?";
pstmt = con.prepareStatement(express);
String c="hello";
pstmt.setString(1, "%"+c+"%");
//参数自动添加单引号，最后的 SQL 语句为：select * from   table where comment like '%hello%'
pstmt.execute();
```

规则5.5　禁止采用未经校验的输入参数来动态构建XPath语句。

说明：和动态构建SQL一样，采用未经校验的输入参数来动态构建XPath语句也会导致注入漏洞（XPath注入）。动态构建XPath语句的例子：

```
public boolean doLogin(String loginID, String password){
    ......
    XPathExpression expr = xpath.compile("//users/user[loginID/text()='"+loginID+"' and
password/text()='"+password+"' ]/firstname/text()");
    ......
}
```

如果有必要采用用户输入数据来构建 XPath，构建前，必须使用“白名单”方式（只允许输入中包含已知的合法字符或组合）对用户输入输入进行严格的校验。以防止攻击者通过构造包含( ) = ` [ ] : , * / 空格等特殊字符的输入数据进行 XPath 注入攻击。

规则5.6　如果服务端代码执行操作系统命令，必须对客户端输入的用于构造命令的参数进行严格的输入校验。

说明：如果服务端代码中使用Runtime.getRuntime().exec(cmd)或ProcessBuilder等执行操作系统命令，那么禁止从客户端获取命令；而且最好不要从客户端获取命令的参数，如果必须从客户获取命令的参数，那么必须采用正则表达式对命令参数进行严格的校验，以防止命令注入（因为，一旦从客户端获取命令或参数，通过;&|<>符号，非常容易构造命令注入，危害系统）。

规则5.7　编写正则表达式对不可信数据进行校验时，避免使用复杂的重复性分组正则表达式对数据进行校验，会产生ReDos攻击。

说明：ReDoS(Regular expression Denial of Service) 正则表达式拒绝服务攻击。开发人员使用了正则表达式来对用户输入的数据进行有效性校验，当编写校验的正则表达式存在缺陷或者不严谨时，攻击者可以构造特殊的字符串来大量消耗服务器的系统资源，造成服务器的服务中断或停止。使用自我重复的重复性分组和替换的重复性分组正则表达式，校验的不可信数据，随着数据长度增长，匹配次数呈指数级别增长。

目前防御手段主要是在程序中避免出现不安全的正则：

- 严格限制用户输入的长度限制。
- 在编写正则的时候，尽量不要使用过于复杂的正则，越复杂越容易有缺陷，且越不容易进行全面的测试。
- 编写正则的时候，尽量减少分组的使用量，使用的越多出现缺陷的可能性越

大。

- 避免动态构造正则（即 new Regex(...)），如果需要构造，也保证不要使用用户的输入来进行动态构造。
- Possessive Match 会阻止正则表达式引擎尝试所有排列可能性，可以提高性能，在满足功能的情况下可以优先考虑使用该模式。

6 文件管理

规则6.1 必须在服务器端采用白名单方式对上传的文件类型进行严格的限制。

说明：必须根据服务器端配置的允许上传的类型，至少对上传文件的扩展名进行校验，拒绝未经允许的文件类型，同时也要避免被注入空字符来绕过扩展名的校验。例如：

空字符 null Java表示 ‘\u0000’，

C中 ‘\0’

URLEncode %00

HTMLEncode 0x00

通常操作系统认为该文件a.txt**null**b.txt 由于空字符的截断，实际操作的文件是a.txt。

规则6.2 禁止以客户端提交的数据作为读/写/上传/下载文件的路径或文件名。

说明：建议对写/上传文件的路径或文件名采用随机方式生成，或将写/上传文件放置在有适当访问许可的专门目录（Web内容目录之外），这些文件不应该具有可执行权限，且他们不应该能够覆盖其他可执行文件。

对读/下载文件采用映射表（例如，用户提交的读文件参数为1，则读取第1个文件file1，参数为2，则读取第2个文件file2）。防止恶意用户构造路径和文件名，实施目录跨越和不安全直接对象引用攻击。

规则6.3 禁止将文件绝对路径发送到客户端。

说明：文件绝对路径发送到客户端会暴露服务端的目录结构信息，有助于攻击者了解系统，为攻击者攻击提供帮助。

规则6.4 禁止存储并执行非可信来源的文件和代码。

说明：一般来说，来自系统之外的文件和代码都是不可信任的，存储和执行非可信来源的文件或代码会带来重大安全隐患，必须禁止执行上传的非可信数据中的应用代码，任何上

传的非可信来源的文件和代码都应放置在Web内容目录之外以防止这些文件被执行。这里所说的执行是指动态页面（如asp、jsp、php）的执行。

规则6.5 对上传的文件大小进行限制。

说明：根据产品的具体要求，需要对上传文件的大小进行限制，避免无谓的消耗网络的带宽和系统的存储空间。

A. 建议在客户端对文件大小进行限制

通过客户端脚本（javascript, active控件等）对上传的文件大小进行限制，可以减少不必要的服务器请求，降低服务器的负载。

B. 必须在服务器端对文件大小进行限制

1. 使用Content-Length作为上传内容的整体大小的判断，Content-Length包含的是Content的整体长度，包含表单数据和上传中的多个文件的大小，只可作为整体内容大小的判断，当Content-Length 大于所设置文件大小的时候，请参考条目3进行。
2. 使用读取上传文件的流的大小来实时判断文件大小。
3. 当上传的文件大小大于限制时需要进行异常处理。

规则6.6 如果服务器端对上传的压缩包进行解压，那么解压前必须检查压缩包中的件路径是否报包含../ 和..\，如果包含则禁止解压，防止目录跨越攻击对上传的文件大小进行限制 。

说明：如果上传的文件是压缩包，那么，程序在从压缩包中提取出的文件路径必须检查压缩包中的文件路径是否报包含../ 和..\，如果包含则禁止解压，以防止攻击者构造压缩包中文件路径使解压文件被释放到目标目录之外的目录（如 Web 内容目录）。

规则6.7 如果服务器端对上传的压缩包进行解压，那么解压前建议对压缩包中各文件大小进行限制。

说明：如果上传的文件是压缩包，那么，程序在从压缩包中解压出文件前，必须先读取并判断文件压缩前的大小是否小于限定的文件大小。Zip 算法的本性就可能会导致 zip 炸弹（zip bomb）的出现，由于极高的压缩率，即使在解压小文件时，比如 ZIP、GIF，以及 gzip 编码的 HTTP 内容，也可能会导致过度的资源消耗。Zip 算法能够产生非常高的压缩比率。例如，一个文件由多行 a 字符和多行 b 字符交替出现构成，对于这样的一个文件可以达到 200：1 以上的压缩比率。使用针对目标压缩算法的输入数据，或者使用更多的输入

数据（无目标的），或者使用其他的压缩方法，甚至可以达到更高的压缩比率。Zip 中任何被提取条目，若解压之后的文件大小超过一定的限制时，拒绝将其解压。具体限制多少，由平台的处理性能所决定。

规则6.8 XML 文件如设计上不需要引入实体，应禁止解析 DTDs；如需要引入内部实体，应禁止解析外部实体，并且限制实体解析个数，避免解析 XML 出现内部实体炸弹攻击。

说明：XML实体注入的防护应该从以下几点入手：

- 防止实体注入最好的方法是禁止解析DTDs。
- 需要使用内部实体，则禁止解析外部实体，外部实体包括外部一般实体和外部参数实体。当攻击者可以篡改XML内容，构造恶意的内部实体，XML解析时会占用服务器内存资源，导致拒绝服务攻击。

XML内部实体是实体的内容已经在Doctype中声明。内部实体格式：<!ENTITY 实体名称 ”实体的值”>。内部实体攻击比较常见的是XML Entity Expansion 攻击，它主要试图通过消耗目标程序的服务器内存资源导致DOS攻击。外部实体攻击和内部实体扩展攻击有不同的防护措施（禁止DTDs解析可以防护外部实体和内部实体攻击）。

7 敏感数据保护

常见的敏感数据有:口令、密钥、数据库连接字符串、证书、会话标识、License、隐私数据（如短消息内容）、授权凭据、个人数据（如姓名、住址、电话等）等。敏感数据的具体范围取决于产品具体的应用场景以及产品的安全策略。

规则7.1 禁止密钥或帐号的口令以明文形式存储在数据库或者文件中。

说明：密钥或帐号的口令必须经过加密存储。例外情况，如果 Web 容器的配置文件中只能以明文方式配置连接数据库的用户名和口令，那么就不用强制遵循该规则，将该配置文件的属性改为只有属主可读写。

规则7.2 禁止在 cookie 、隐藏域中以明文形式存储敏感数据。

说明 1：cookie 信息容易被窃取，尽量不要在 cookie 中存储敏感数据；如果条件限制必须使用 cookie 存储敏感信息时，必须先对敏感信息加密再存储到 cookie。会话 cookie 中的 SessionID 除外。

说明 2：隐藏域中明文敏感信息，可以通过查看页面源码来获取。

规则7.3 禁止在日志中记录明文的敏感数据。

说明：禁止在日志中记录明文的敏感数据（如口令、会话标识等）， 防止敏感信息泄漏。

规则7.4 禁止带有敏感数据的 Web 页面缓存。

说明：带有敏感数据的 Web 页面都应该禁止缓存，以防止敏感信息泄漏或通过代理服务器上网的用户数据互窜问题。特定条件下，即便指定了”Cache-Control: no-cache”头域，浏览器仍会执行缓存动作。建议应考虑使用”no-store”指令。

规则7.5 对存储在服务器上的敏感数据实施访问权限控制。

说明：基于操作系统和数据库的访问控制机制，对服务器上存储的敏感数据实施严格的访问权限保护，即对于敏感数据文件只有属主才能够访问，对于存储在数据库中的数据，只有已授权的用户可以访问。

规则7.6 带有敏感数据的表单必须使用 HTTP-POST 方法提交。

说明：禁止使用 HTTP-GET 方法提交带有敏感数据的表单（form）。通过 HTTP-POST 提交带有敏感数据的表单可以有效阻止跨站请求伪造攻击，并且可以避免带有敏感数据的 URL 被记录于服务端日志、代理日志、浏览器历史而造成的信息泄露。

规则7.7 在客户端和服务器间传递敏感数据时，必须保证数据的机密性。

说明：如果在客户端和服务器间传递如帐号、口令等敏感数据，必须保证数据的机密性，推荐使用带服务器端证书的 SSL。

规则7.8 禁止在 URL 中携带会话标识。

说明：由于浏览器会保存 URL 历史记录，Web 服务器和代理服务器的日志也都可能保存 URL 信息，如果 URL 中携带会话标识（如 jsessionid），则容易造成会话标识泄露，一旦该会话标识还在其生命有效期，则恶意用户可以冒充受害用户访问 Web 应用系统。

8 防范 CSRF

跨站请求伪造（CSRF）是一种挟制终端用户在当前已登录的 Web 应用程序上执行非本意的操作的攻击方法。攻击者可以迫使用户去执行攻击者预先设置的操作，例如，如果用户登录网络银行去查看其存款余额，他没有退出网络银行系统就去浏览自己喜欢的论坛，如果攻击者在论坛中精心构造了一个恶意的链接并诱使该用户点击了该链接，那么该用户在网络银行账户中的资金就有可能被转移到攻击者指定的账号中。当 CSRF 针对普通用户发

动攻击时，将对终端用户的数据和操作指令构成严重的威胁；当受攻击的终端用户具有管理员帐号的时候，CSRF 攻击将危及整个 Web 应用程序。

规则8.1　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）。如使用动态令牌、随机短信、验证码进行 CSRF 防范，需要保证长度不小于 4。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。

规则8.2 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。

规则8.3 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。

规则8.4 仅在业务需要时，开启特定网页的跨源资源共享，并对来源/域进行限制。

说明： 出于安全性的考虑，浏览器的同源策略，拒绝跨站点的访问请求，限制运行在用户代理的 Web 应用通过 Ajax 或者其他机制从另一个站点访问资源、获取数据。跨源资源共享（Cross-Origin Resource Sharing）提供一种机制，允许客户端（如浏览器）对非源站点的资源发出访问请求。跨源资源共享（CORS）通过使用自定义的 HTTP 响应头部（HTTP Response Header），通知浏览器资源可能被哪些跨源站点以何种 HTTP 方法获得。

如果不是业务需要，禁止开启跨源资源共享（CORS）；如果业务需要，开启特定网页的跨源资源共享，并对来源/域进行限制。

如果业务需要，开启特定网页的跨源资源共享，并对来源/域进行限制，方法如下：

Java： Response.AddHeader("Access-Control-Allow-Origin", "http://www.1688hot.com:80");

html： <meta http-equiv="Access-Control-Allow-Origin" content="http://www.1688hot.com:80">

规则8.5 如果产品存在跨域复杂请求（CORS），Preflight请求的返回结果需要对域、允许跨域的方法，允许跨域扩展的HTTP头进行限制。

说明：服务器对 Preflight 请求的结果中对允许的访问域，方法，扩展的 HTTP 头进行限制

1. Access-Control-Allow-Origin：服务器所设置的允许域的白名单，不允许使用*来表示任意跨域请求
2. Access-Control-Allow-Methods：服务器所设置的允许跨域的请求方法白名单列表，对多个方法使用逗号作为分割符

3. Access-Control-Allow-Headers：服务器所设置的允许跨域的请求中扩展的 HTTP 头白列表名单，对多个头使用逗号作为分割符
4. Access-Control-Max-Age：指定本次预检请求的有效期，单位为秒，建议设置大小小于会话生命周期

规则8.6 不信任未经身份验证的跨域请求。

说明：由于 HTTP 头容易被伪造，所以不能信任未经身份验证的跨域请求。如果一些重要的功能需要暴露或者返回敏感信息，必须验证会话标识是否合法、用户是否已经被授权执行该操作。

9 防范 XSS

规则9.1 若输出到客户端或者解释器的数据来自不可信的数据源，则须对该数据进行相应的编码或转义，防范 XSS 攻击。

说明：跨站点脚本（XSS）是一种允许攻击者在用户的浏览器中执行恶意脚本的脚本注入式攻击。XSS 主要分为反射型和存储型。

反射型 XSS，是指发生请求时，XSS 代码出现在请求 URL 中，作为参数提交到服务器，服务器解析并响应。响应结果中包含 XSS 代码，最后浏览器解析并执行。

存储型 XSS，主要是将 XSS 代码发送到服务器，由服务器保存起来，然后在下次请求页面的时候，服务器返回存储的 XSS 代码，最后浏览器解析并执行。

输出编码和转义是指将输出数据中的一些特殊字符转换成安全的形式，使得目标解释器不会将其当作语法符号或指令，以防止原本预期的语义被更改，避免跨站脚本攻击与各种类型的注入攻击。

10 安全加密

规则10.1 禁止使用非标准和不安全的密码算法，推荐使用强密码算法

说明：私有的、非标准的密码算法不应该在 Welink 产品中使用。一方面，非密码学专业人员设计的密码算法，其安全强度难以保证。另一方面，这类算法在技术上也未经业界分析验证，有可能存在未知的缺陷。另外随着密码技术的发展以及计算能力的提升，一些密码算法已不再适合现今的安全领域，例如 MD5 算法和 DES 算法。下表是华为公司推荐使用

的安全加密算法：

用途		常见不安全密码算法	可遗留使用的密码算法	推荐使用的强密码算法
对称加密	分组加密	Blowfish, DES, DESX, RC2, Skipjack, 2TDEA, TEA、3DES(未遵循规则**错误!未找到引用源。**)	3DES(遵循规则**错误!未找到引用源。**)	AES-GCM(≥128 bits)
	流加密	SEAL, CYLINK_MEK, RC4 (<128 bits)	RC4(≥128 bits)*	AES-CTR(≥128 bits) AES-OFB(≥128 bits) chacha20
哈希算法		SHA0, MD2, MD4, MD5, RIPEMD, RIPEMD-128	SHA-1*	SHA256或以上 SHA3
非对称加密		RSA (< 2048 bits) ECIES (< 224bits)	RSA* (<3072 bits) ECIES (<256bits)	RSA(≥3072 bits) ECIES (≥256 bits)
数字签名		RSA (< 2048 bits) DSA (同X9.42标准DH) ECDSA(<224bits)	RSA(<3072 bits) DSA(同X9.42标准DH) ECDSA(<256 bits)	RSA(≥3072 bits) DSA(同X9.42标准DH) ECDSA(≥256 bits)
密钥协商	DH	L<2048 bits N*<224 bits	L<3072 bits N<256 bits	L≥3072 bits N≥256 bits
	ECDH	ECDH(< 224bits)	ECDH(<256bits)	ECDH(≥256 bits)

规则10.2 密码算法中使用到的随机数必须是密码学意义上的安全随机数

说明：随机数用于生成 IV、盐值、密钥等，都属于密码算法用途。不安全的随机数使得密钥、IV 等可被全部或部分预测。密码学意义上的安全随机数，要求必须保证其不可预测性。

产品可以直接使用真随机数产生器产生的随机数。无法获取足够的真随机数时，也可以使用真随机数产生器产生的少量随机数做种子，输入密码学安全的伪随机数产生器产生大量密码学安全伪随机数。已知的可供产品使用的密码学安全的伪随机数产生器包括：

- OpenSSL库的RAND_bytes
- OpenSSL FIPS模块中实现的DRBG（NIST SP 800-90A标准）
- JDK的java.security.SecureRandom（从JKD 9起支持NIST SP 800-90A标准）

规则10.3 使用分组密码算法时，禁止使用 ECB 模式，应优先选择 GCM 模式

说明：分组密码算法有多种工作模式，不同工作模式有特定的适用场景。ECB 模式对于同样的明文块会生成相同的密文块，不能提供严格的数据保密性，不能抵抗替换攻击，攻击者可以调换加密块的顺序而不被发现。GCM 模式是目前业界推荐使用的对称加密算法工作模式，产品应该根据具体应用场景，存在 GCM 加密模式时优先使用 GCM 加密模式。

规则10.4 使用 RSA 算法进行加密操作时，应使用 OAEP 填充方式

说明：对数据进行填充要非常谨慎，因为一些有经验的黑客有可能从中找到一些线索。早期的 PKCS#1 填充标准就曾受到一种自适应选择明文攻击的威胁。RSA 实验室在 PKCS#1 V1.5 以后的标准中增加了 OAEP（Optimal Asymmetric Encryption Padding）填充模式，可以有效阻止这类威胁。一般算法库提供的接口函数中都会有填充方式选择参数，如 OpenSSL 中为 RSA_PKCS1_OAEP_PADDING。

规则10.5 在同时需要对数据进行对称加密和数字签名时，使用先签名后加密的方式

说明：先签名后加密是指先对消息进行签名，然后对消息的签名值和消息一起进行加密。如果采用先加密后签名的方式，接收方只能知道该消息是由签名者发送过来的，但并不能确定签名者是否是该消息的创建者。比如在发送一个认证凭据时采用先加密后签名的方式，消息在发送过程中就有可能被第三方截获并将认证凭据密文的签名值修改为自己的签名，然后发送给接收方。第三方就有可能在不需知道认证凭据的情况下通过这种方式来通过认证获取权限。采用先签名后加密方式可以避免这类问题的发生，因为只有在知道消息明文的情况下才能对其进行签名。

规则10.6 禁止使用 SSL2.0、SSL3.0、TLS1.0 协议，建议使用 TLS1.2 及以上版本

说明：SSL2.0 和 SSL3.0 协议因存在很多安全问题已经被 IETF 禁用。TLS1.0 协议支持的所有加密套件中，对称加密算法仅支持 RC4 算法和分组加密算法的 CBC 模式，RC4 已经

公认不安全并被 IETF 在 TLS 所有版本协议中禁用，而对称算法的 CBC 模式存在 IV 可预测的问题，从而容易受到 BEAST 攻击。因此，禁止使用 TLS1.0 协议。

TLS1.1 协议目前虽然没有明显安全问题，本规范中目前仍可使用，但该版本相对 TLS1.0 在算法套件上没有增强，且用于密钥派生的 PRF 算法基于 MD5 和 SHA1 算法。当前 IETF 已经在计划废止 TLS1.0 和 TLS1.1，强烈建议使用 TLS1.2 及以上版本。