API 安全开发指南

V 2.1

目录

1 目的 3

2 适用对象 3

3 安全开发基础 3

3.1 安全设计 3

3.2 功能和策略安全 4

3.2.1 口令和认证 4

3.2.2 授权管理 4

3.2.3 访问控制 4

3.2.4 日志审计 5

3.2.5 通信安全 5

3.2.6 数据保护 6

3.2.7 容错设计 6

附录：API 安全开发 CHECKLIST 7

1 目的

为尽量避免 API 在设计、开发和运维过程中产生安全缺陷和漏洞，以及降低 API 在遭受攻击时所带来的风险和影响，规范提出 API 开发、运维应达到的安全要求和需要，并提出安全开发的方法、流程和处理建议，保障业务的安全性。

2 适用对象

1、API 开发人员

2、API 运维人员

3、API 安全负责人

3 安全开发基础

3.1 安全设计

在 API 开发之前，应考虑安全问题，并进行业务安全设计。安全设计应满足以下条件：

◆ **政策**

为保障业务符合可用性和灾难恢复要求，API 应遵循公司和国家、国际、行业标准和指南，避免因为未能满足特定安全开发要求而导致的业务安全问题。

◆ **原则**

为保证 API 在可用性、可靠性、容灾性、完整性、机密性上的要求，开发安全设计应满足以下原则：

权限最小化	对用户进行权限分级化管理。在创建一个新的用户时，默认的权限应当是最小的
纵深防御	对于需要保护的数据和应用，应部署冗余安全措施，即当防御失效的时候，应当有其他防御可使用。在保证可用性和成本的情况下使用纵深防御
简单设计	复杂的设计更容易出现问题，尽量使功能的实现简单化
故障安全化	当业务出现故障警告时，应考虑安全显示，并自动关闭有问题的程序

保护最薄弱的环节	识别出业务 API 中最易受攻击的数据和 API，如账号登录、忘记密码、文件上传、短信验证码发送等，并实施保护
对外开放最小化	对于非面向大众用户的 API，尽量使用 IP 和端口、账号白名单对访问、输入操作进行默认拒绝。尽量不使用黑名单
保护数据和隐私	识别并保护用户的数据和隐私资料，并对 API 中流动的数据进行脱敏、对访问异常进行检测
尽量多的独立单元	功能模块尽量分层、分离，使攻击面最小化
策略一致	API 的访问授权、安全策略、数据脱敏保持一致，保持每次访问都进行安全检查，不同 API 针对同样的敏感数据脱敏策略是一致的
公开设计	不假设 API 中存在隐藏的设计，比如安全策略、密钥不会被外界发现

3.2 功能和策略安全

安全功能是 API 本身应具备的功能，例如需要添加验证码、需要限制频率、需要设置白名单、确保安全登录和退出功能等等。在业务需求形成的阶段，应考虑 API 中应具备的安全功能需求，从而制定开发方案。

3.2.1 口令和认证

登录认证功能尽量做到标准统一、入口统一，避免认证绕过、失效、被破解等问题。

3.2.2 授权管理

API 要对用户访问权限和数据权限进行检查，避免因为授权逻辑漏洞而造成的越权操作和数据泄漏。尽量避免在公网上暴露管理类系统的 API，如果必须在公网上暴露，尽量采用 IP 白名单的方式来进行限制。

3.2.3 访问控制

API 应当对会话和连接进行访问控制，避免用户访问的泄露、会话劫持、拒绝服务攻击等。可采用如下的措施：

- 应设定用户会话空闲时间，超时自动退出程序；系统应有“退出”和“注销功能”，允许用户终止当前会话
- 限制系统的最大并发数、单个用户的多重并发会话进行限制、对一个时间段内可能的并发会话连接数进行限制
- 设置严格的目录访问权限，防止未授权的访问，对访问的出错信息要进行转换处理，避免直接暴露后端系统信息
- 不使用目录列表浏览，防止重要数据被攻击者发现或未授权下载
- 对于文件上传功能，应做到可以通过白名单、校验文件头的方式控制可以上传的文件类型，控制上传文件的大小和上传路径，同时对上传路径进行读写权限控制

3.2.4 日志审计

API 访问要保存访问日志，保证操作不被抵赖，并对日志进行操作行为的安全分析和审计，安全日志审计可参考如下几个方面：

- 日志至少记录访问者的 IP 地址、终端信息、认证账号、认证结果、操作时间、操作类型、操作结果等信息。
- 如交易日志和涉及到敏感数据的重要日志、如涉及计费信息、身份证、密码等数据进行加密处理，不允许删除、篡改、覆盖
- 在服务器端记录 API 错误信息日志
- 对日志的访问权限进行访问控制策略
- 对重要日志部署专门的采集、备份、管理中心，定期对日志进行安全分析、生成审计报表

3.2.5 通信安全

API 对于重要数据及指令的传输应进行通信加密和校验，防止包重放攻击和信息的泄露、篡改。可以采用的方法有：

- 使用安全的加密算法保护客户端和 API 之间的关键连接，如 SSL/TLS，SSL/TLS 使用新的版本，并取消对低版本的支持
- 对通信进行完整性校验。使用强度更强的散列算法会更安全，如 SHA256

3.2.6 数据保护

API 应做到对传输的数据和关键信息进行加密、脱敏处理，避免因为 API 被爬取带来数据泄露：

- 根据业务的需要，对敏感信息如用户的银行账号、身份证号码、电话号码、交易数据等信息进行加密或脱敏传输，加密算法应采用不可逆的加密算法
- 采用密码技术对数据进行完整性校验和机密性保护。对重要执行文件，通过对确定文件产生的散例值 来验证文件信息的有效性和真实性
- 针对数据敏感的 API，需要结合 IP、终端信息、身份鉴别、访问控制等多方面特征和控制策略来构建实时动态的安全策略检测异常的数据访问，避免数据泄漏

3.2.7 容错设计

容错设计是指 API 在出现输入错误、运行异常、遇到攻击的时候，也能保证系统的安全性：

- API 需要对输入的参数和运行的逻辑进行检查，确保能够防御如下攻击：SQL 注入漏洞攻击、XSS 漏洞攻击、CSRF 漏洞攻击、命令注入漏洞攻击、重放攻击漏洞的能力等
- 针对 API 的运行的服务器，要正确配置服务器和应用，避免因为配置错误造成的安全问题
- API 不要放到线上运营系统来进行测试，避免遭受黑客攻击

附录：API 安全开发 checklist

安全开发	安全功能要求项
口令和认证	登录 API 尽量统一，保证公司只有一套登录 API
	用户身份、访问权限和数据权限进行校验
	需要有实时动态的安全策略防止撞库、暴力破解、恶意注册
	使用安全性更强的验证码，如行为验证码
	设定密码时，需对密码强度进行检查： 1） 密码长度至少为 8 位 2) 密码需包含大/小写字母、数字、特殊符号 3) 设定密码时，避免用户名与密码一致
	避免明文传输密码： 1） 在 API 通信过程中，不使用明文进行用户密码传输，必须加密后再进行传输 2) 建议使用非对称加密算法或者哈希算法对密码进行加密，避免密钥泄露导致加密密码被还原
	避免在 URL 中传输密码或凭证： 1） 密码或凭证信息不要以参数等形式在 URL 中传输，避免关键信息被浏览器访问记录或后端访问日志存储导致的数据泄露风险 2） 建议 API 调用采用 POST 方法，通过请求体进行密码或凭证等敏感参数的传递 3） 建议 API 通信采用 HTTPS 协议
	防止账号被盗： 1） 在涉及到用户名、密码验证的 API 接口中，当密码输入错误超过一定次数，建议启用二次验证（如滑块验证码、短信验证码等）加强安全，防止密码爆破行为 2） 当用户常用登录设备或常用登陆地发生变化，建议启用二次验证（如滑块验证码、短信验证码等），防止密码爆破行为
	选择合适的认证算法： 1） Authorization 不要使用 Basic 认证方式，Basic 认证方式将账号及密码进行 BASE64 编码后传输，该编码方式通过解码即可得到明文账号密码，存在用户名和密码同时泄露的风险

	2） 建议采用 Digest Authentication、OAuth Authentication、Token Authentication 等无法反解的认证方式来完成认证
	设置凭证失效： 1） 在账号登出或注销后，需要将已经验证通过的凭证设置为失效 2） 在账号完成密码修改后，需要将已经验证通过的凭证设置为失效 3） 长时间不进行操作，需要将已经验证通过的凭证设置为失效
授权	对于非完全开放的资源，尤其是访问用户的资源或者受限资源，针对访问行为都需要进行授权验证
	对于不需要授权访问的 API 接口，可以通过白名单方式来限定 API 接口的开放范围
	针对用户对系统资源的访问行为，API 需检查该用户是否具备该资源的访问权限，用户不允许访问其他用户的数据和功能，也不允许访问其他角色（尤其是管理角色）的数据和功能
	针对用户身份标识（如 UserId）需使用不被预测的规则生成，避免关键数据被枚举导致的数据泄露
后台管理 API	避免这类 API 暴露在公网
	如果必须在公网上暴露，尽量使用 IP 白名单限制登录
数据保护	在客户端和后端 API 都对用户输入的数据进行格式和类型的校验、转义，控制恶意输入
	针对返回敏感数据的 API，避免使用用户 ID 等易猜测的明文信息作为 API 的输入，避免数据被遍历爬取
	敏感数据查询 API 保护： 1） 敏感数据的查询 API，应集中管理且严格控制权限，避免分散加大管理难度 2） 敏感数据的查询 API，必须具备访问频率控制能力，避免数据被批量盗取 3） 敏感数据的查询 API，对重点敏感数据要脱敏返回
	避免明文传输敏感数据，针对敏感数据要进行脱敏： 敏感信息脱敏展示时，应当符合数据脱敏规范，避免脱敏不规范导致数据可被枚举还原，可参考《JR/T 0223 金融数据安全 数据生命周期安全规范》的字段脱敏规范，常见的几种数据类型的脱敏规范如下： 1） 身份证：建议只显示第一位和最后一位字符，如 3****************1 2） 手机号：建议隐藏中间 6 位数字，如 134******48 3） 银行卡：建议只显示最后 4 位字符，如**********8639

	4) 工作地址/家庭地址：建议只显示到区一级
文件保护	通过白名单的方式控制可以上传的文件类型
	通过校验文件头的方式来控制上传的文件类型
	控制上传文件的大小和上传路径，同时对上传路径进行读写权限控制，上传目录禁止执行脚本；争取使用独立的存储设备对上传文件进行存储
	避免文件目录遍历： 1) 系统需禁用目录浏览，如系统开启了目录浏览功能，则存在 Web 站点结构暴露的风险，攻击者可利用对站点进行定向攻击 2) 系统需禁用默认访问，只对指定目录开放访问权限，对敏感文件进行严格的权限控制以及文件加密，避免数据被恶意获取 3) 系统需禁止用户对目录配置文件（.htaccess）进行修改
	建议对外部输入的文件路径进行合法性校验，过滤掉输入内容中可导致路径穿越的字符，如（../ ..\等）

容错设计	**防御 SQL 注入攻击：**
	● **对外部输入进行安全过滤**
	1) 默认不信任任何外部输入，当外部输入采用动态拼接 SQL 语句输入时，系统必须对输入进行安全过滤
	2) 建议系统使用白名单方式进行严格的输入合法性检查，禁止任何未通过合法性检查的输入
	● **对外部输入进行转义**
	1) 在输入内容包含了数据库系统的元字符时，需对输入内容进行转义操作，避免输入内容与数据库系统的元字符冲突，有效防止 SQL 注入攻击
	2) 建议使用专门的转义组件来对输入内容进行转义，需选择经过严格安全测试的转义组件产品
	● **使用参数化查询**
	1) 建议使用参数化查询方式来进行数据查询及操作，此方式预先定义 SQL 语句查询结构，用户输入的内容均作为参数传入，数据库可清晰的区分出 SQL 指令和用户输入的数据，从而阻止对 SQL 查询结构的篡改，有效防止 SQL 注入攻击。举例来说，假设攻击者在输入 UserID 时键入“ 1'or '1'='1 ”，攻击也不会生效，因为数据库会将“ 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
	2) 参数化查询的 Java 代码示例如下：
	`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();
```

防止 XSS 攻击：

1） 限制、判断客户端提交的数据类型、有效性，对输入进行校验

2) 对数据的输出进行编码转义，防止恶意信息输出利用

防御 CSRF 攻击：

- **对 Referer 字段进行校验**

1） 建议使用白名单方式对 Referer 字段进行合法性检查，拒绝任何未通过 Referer 合法性检查的请求

- **使用一次性令牌**

1） 建议系统使用一次性令牌，后端在令牌验证通过后再进行业务逻辑的处理

2） 令牌需在完成工作后立即执行销毁操作，迫使攻击者无法重复使用单一令牌来发起攻击请求

3） 可以结合当前时间来生成令牌 ID，保证令牌 ID 的唯一性

- **添加验证码**

1) 针对重要的操作，可增加随机验证码检测

防止程序导致的业务和服务器信息泄露：

1) 禁止程序在错误提示中包含详细信息，不向用户显示调试信息，使用自定义友好界面代替

防止语义 URL 攻击：

1) 针对 API 有返回敏感数据的情况，不使用用户名称、ID、手机号等易于猜测的信息作为 API 的参数，避免遍历爬取

2) 对需要权限访问的 API 进行身份验证审查

防止命令注入：

- **对外部输入进行安全过滤**

1) 默认不信任任何外部输入，当外部输入采用动态拼接进行命令执行语句输入时，对输入进行安全过滤

2) 使用白名单方式来进行严格的输入合法性检查，禁止任何未通过合法性检查的输入

3) 对特殊字符进行转义（|、&、；等），防止非法命令执行

4) 利用 可依赖的变量在执行过程中构建绝对路径，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

5) 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 ● **避免直接调用系统命令** 1) 避免使用直接调用系统命令的方式来执行代码，而使用库函数或者自定义函数来实现同等功能 2) 如无法避免，则建议在最小权限原则的基础上，严格控制通过接口可执行的系统命令范围
	防御重放攻击： 1) 对数据进行会话验证等用户访问有效性验证 2) 设置会话超时
	防御任意 URL 跳转攻击： 1) 对于 API 是实现输入参数是 url 时进行跳转的逻辑，需要对 url 参数进行验证，尽量使用白名单 2) 尽量跳转是公司自己的域名，避免任意 url 输入都可以进行跳转
安全审计	在服务器端日志记录客户端的 IP 地址和主机名、认证者身份、认证结果、操作时间、操作类型、操作结果；尽量包含足够详细的信息能来识别恶意的攻击者，包括所有失败的身份验证尝试、拒绝访问和输入验证错误等
	重要日志如交易日志、计费信息，进行加密或脱敏处理，不允许删除、篡改、覆盖
	在服务器端记录错误日志
代码安全	禁止将敏感信息直接写入源代码中，这些敏感信息包括：API tokens、密码、数据库凭证、数据库的说明、保密的日志等
	禁止将敏感信息存放到代码管理平台中，例如公司内部的 Gitlab、外网的 GitHub
	代码上线前，需要进行代码混淆和加固，并去除所有注释信息