1. API安全规范检查表

（阅前须知：1）每条检查项后面中括号中的数字代表着其在“02-附件：文献筛选与编码及检查项5W要素补充表”中的“5W要素补充”子表中的序号，方便阅读者索引获取检查项具体的上下文信息；2）每条检查项后面标明了其是“思想指导”还是“具体检查项”，思想指导指的是该检查项抽象层级较高，是方法论层级的指导，而“具体检查项”则指的是其是有落地可能性的，针对具体问题的安全检查项；3）有些检查项下有下划线，这指的是该检查项缺乏5W相关的上下文信息）

1. 访问控制
2. 经过认证和授权的用户或应用程序才可以访问API。【1】（思想指导）
   1. 身份验证技术
      1. 身份验证方法选型
         1. 采用标准认证流程来防止引入漏洞。【6】（思想指导）
         2. 不同端点采用不同的认证机制 ，确保生产环境的认证端点有防暴力破解保护。检查端点验证授权输入，确保访问API端点时不能绕过网关。【7】（思想指导）
         3. 使用安全的身份验证机制，如基于令牌的身份认证方案（Oauth、Oauth2）或结合了其它的安全机制（API密钥、密码策略等）的身份验证机制，确保只有经过授权的用户才可以访问资源。即使在安全性要求较低的场景，也应避免使用HTTP基本身份认证与HTTP摘要认证，后两者均存在安全风险(尽即使HTTP摘要身份验证比发送明文的基本接入认证更安全，此时可以使用仅有两个角色的简化的Oauth2框架 。【2】（具体检查项）
         4. 包括laas平台在内的服务端与客户端应该避免使用单因素身份认证(如使用API密钥作为唯一的认证手段)，并提供多因素身份认证或其它被认为比基于密码的身份验证更安全的方案，以防止攻击者利用泄露的部分信息进行非法操并防止CSRF攻击。【3】（具体检查项）
         5. 在服务端存储和检查用户会话的前提下，才可以使用基于cookie的身份验证。【4】（具体检查项）
         6. 整合OpenID Connect作为身份验证框架，管理和验证用户的身份。【5】（具体检查项）
      2. Oauth
         1. 使用OAuth认证技术。在使用OAuth2时必须有状态参数。【9】（思想指导）
         2. 将不同的客户端应用的Oauth进程聚合为一个统一的客户端应用与流程。 【10】（思想指导）
         3. 通过基于生物特征的身份认证增加Oauth的安全性。【8】（具体检查项）
         4. 使用SDN API时选择将授权服务器与资源服务器（SDN控制器）部署在一台服务器，而不是如Oauth2规范要求的分离。【11】（具体检查项）
         5. 针对物联网对互操作性的要求，使用Oauth时，ROs（资源所有者）、客户端、授权服务器、资源服务器间必须相互进行身份验证，并提供可信的验证方案（使用TLS和数字签名等）。【12】（具体检查项）
      3. 身份验证过程加密
         1. 仅使用安全的HTTPS，TLS/SSL实现基于API密钥的身份验证方案。【13】（思想指导）
         2. 使用包含API用户注册、SSL和强密码保护的基本身份验证。【14】（思想指导）
         3. 客户端需要加密凭证，并在确认授权服务器的身份后进行后续验证。 【15】（思想指导）
      4. 身份验证凭证设计与管理
         1. 采用强大的密码策略、限制登录尝试次数以及实施账户锁定机制等措施，降低暴力破解密码的风险。【16】（思想指导）
         2. 应用程序可以为开发版本和运行版本的API生成不同的密钥，以降低密钥暴露的风险。【19】（思想指导）
         3. 使用非对称密钥，为不同的API密钥配置不同的权限，不要在代码中硬编码来自第三方SDK的密钥并采取密钥轮换策略防止窃听或截取。确保API密钥的安全存储并监控其使用。【21】（思想指导）
         4. 使用基于图像处理的验证码服务防止验证过程出错。【17】（具体检查项）
         5. 设置验证码与二维码的过期时间，防止验证码盗用后的攻击。【18】（具体检查项）
         6. 在消息验证码（MAC）生成的过程中，可以将API密钥与一些请求参数串联生成密钥，将其置入散列算法中生成安全的密钥，并且可以结合更安全的MAC算法（如HMAC-SHA1）。【20】（具体检查项）
         7. 授权服务器需要监控MROPC(使用多个资源所有者的密码凭证来交换一个身份验证令牌的流程)，限制每个用户短时间内MROPC请求的数量，防止攻击者通过MROPC猜测单个资源密码凭证的风险。【22】（具体检查项）
         8. 实施最大重试和监狱安全机制，以防止密码猜测攻击和其他认证尝试的滥用。【23】（具体检查项）
         9. 在移动端OTP(One-Time Password)基于手机验证时，调用生成OTP或重新发送OTP API时，不要在响应中返回OTP。【24】（具体检查项）
   2. 授权
      1. 授权凭证设计与管理
         1. 授权令牌
            1. 在验证令牌时，不仅要验证令牌的有效性，同时也要检查令牌对应用户组的权限范围。授权服务器需要具有主动令牌刷新机制，并为令牌设置合理（建议使用短期）的到期时间，避免过度授权。【25】（思想指导）
            2. 使用JWT认证技术， JWT提供了一种有效的方式来确保数据在传输过程中的安全性，并支持无状态认证。【29】（思想指导）
            3. 使用SDN API时选择存储在SDN控制器上的不记名令牌作为访问令牌设计。【26】（具体检查项）
            4. 使用附加的CSRFToken，在请求头中携带特殊的CSRF安全令牌（:X-Csrf-Token: token\_value）以防止CSRF攻击。【27】（具体检查项）
            5. JWT（Json Web Token）容易受到以利用报头算法和解码有效载荷为重点的用户身份验证失败的攻击，当对安全要求较高时而对性能要求较低时，考虑采用PASETO(平台无关的安全令牌)而不是JWT。【28】（具体检查项）
            6. 针对物联网对互操作性的要求，使用Oauth时，授权服务器刷新令牌时需要通知资源所有者，防止新的令牌被滥用。 【30】（具体检查项）
            7. 针对物联网对互操作性的要求，使用Oauth时，IAT（互操作性授权令牌）需要与客户端绑定，并设置较短的过期时间。 【31】（具体检查项）
            8. 针对物联网对互操作性的要求，使用Oauth时，授权服务器需要使用加密安全的随机或伪随机数生成器发布授权码和用户身份令牌。【32】（具体检查项）
            9. 使用OAuth2.0时，对不同用户发放不同的访问令牌。当共享用户的权限失效时，应主动注销共享用户的令牌。 【33】（具体检查项）
            10. 检查通过邮件或短信发送的重置密码令牌的随机性。【34】（具体检查项）
            11. 使用中央OAuth服务器发放访问或刷新令牌。【35】（具体检查项）
         2. 其它授权凭证
            1. 提供密钥版本控制功能，允许开发人员和使用者更新会话密钥、公钥和私钥，并配套相应的权限控制策略。【36】（思想指导）
            2. 针对物联网对互操作性的要求，使用Oauth时，在授权码的交换过程中使用PKCE（Proof Key for Code Exchange）。【37】（具体检查项）
            3. 登出或重置密码后销毁会话标识符。【38】（具体检查项）
      2. 避免过度授权
         1. 提供细粒度的访问控制策略，避免定义粗粒度的权限，粗粒度的权限可能导致用户或应用被过度授权，进而产生安全问题。【39】（思想指导）
         2. 使用适当权限的账号访问数据库。确保使用的账号权限有限，并被管理员所监控与限制。【44】（思想指导）
         3. 限制账号权限并进行白名单验证，可以预防SQL注入。【45】（思想指导）
         4. 实施用户层次的应用功能访问限制。避免恶意攻击者仿造用户请求。 【46】（思想指导）
         5. 开发文档应明确各方法的权限，避免过度授权与不明确的授权。【47】（思想指导）
         6. 静态资源的授权可能与动态内容不同，确保两者都有适当的安全措施。【52】（思想指导）
         7. 实施基于角色的授权策略（如基于角色的访问控制（RBAC）），确保用户只能执行其角色许可的操作，防止未经授权的活动。【63】（思想指导）
         8. 避免使用通配符“\*”，其会导致过度授权问题。【40】（具体检查项）
         9. 服务端需限制上传的文件或目录执行操作的权限。上传的目录不应具有任何的执行操作权限，避免攻击者上传恶意脚本文件进行安全攻击。【41】（具体检查项）
         10. 服务端仅可使用白名单机制验证文件名及扩展类型，黑名单的方式并非完全可靠。【42】（具体检查项）
         11. 多参数的API在每一个参数组合下都应有授权策略保障，同时切断无关参数之间的联系，比如将其分为两个函数等，避免无关参数间互相影响访问权限。【43】（具体检查项）
         12. 授权服务器需要具有主动令牌刷新机制，并且设置合理的令牌刷新周期。【42】（具体检查项）
         13. 正确配置CORS策略，限制哪些域可以访问API，防止跨域攻击。 【48】（具体检查项）
         14. 父资源的子资源不可从另一个父资源访问，防止攻击者使用未经验证的父对象标识符访问子资源。【49】（具体检查项）
         15. 在一个用户命名空间创建的资源不能被其它用户命名空间访问 ，防止攻击者伪造身份令牌，在属于其它用户的资源空间内进行违规操作。【50】（具体检查项）
         16. 避免方法名和实际行为不匹配，在这种情况下，API的最小权限集可能会被模糊，并且导致过度授权的情况 。【51】（具体检查项）
      3. 防火墙与网关配置
         1. 在系统中加入API网关组件，可以对请求进行批量处理并应用授权策略；将防火墙加入API网关还可以带来额外功能如流量速率限制、防止注入攻击等 。使用API网关强制执行访问控制，访问API端点时不能绕过网关。【54】（思想指导）
         2. 配置网络防火墙来过滤恶意流量，并使用SSL协议对数据传输进行加密，保护API免受网络层面的攻击。【55】（思想指导）
         3. 使用防火墙技术，阻止或过滤危险的针对数据库的访问，如果服务提供商使用无效证书，防火墙禁止其进行任何操作。如果服务提供商是SP1（Windows 7 Service Pack 1 ），防火墙禁止其访问位置数据。如果服务提供商已经收集了电话号码，阻止它获取邮政地址，从而避免用户隐私信息泄露【53】（具体检查项）
      4. 其它
         1. 对用户、应用、设备均进行鉴权，通过鉴权确定用户可以访问的资源。 【56】（思想指导）
         2. 设计并使用集中式访问控制策略。【58】（思想指导）
         3. 建立跨组织的授权管理机制。 【59】（思想指导）
         4. 可以从API功能文档中生成授权策略，再辅以人工指定授权策略。【60】（思想指导）
         5. 可以从应用程序的OAS(Option-Adjusted Spread)中学习用户访问模式和功能，使用强化学习(RL)系统的解决方案可以通过训练来测试应用程序的访问控制策略并发现缺陷，并解决这些缺陷。【61】（思想指导）
         6. 防止通过参数污染绕过安全控制，引入BOLA(broken-object-level-authorization)漏洞。【64】（思想指导）
         7. 授权机制需要健壮，可考虑结合多个框架和方法，比如结合Oauth2框架、OpenID连接（OIDC）、TLS身份验证，JWT短期令牌授权、单点登陆（SSO）等。【57】（具体检查项）
         8. 避免通过大量分配（可能是指资源或请求）来绕过安全机制并执行非法操作（如更新密码）。【62】（具体检查项）
3. 数据传输保护
   1. 数据传输保护
      1. 数据传输加密
         1. 使用一定的措施防止密码或数据明文传输与存储。【69】（思想指导）
         2. 加密授权服务器通信，防止授权信息泄露。【70】（思想指导）
         3. 使用私钥对访问令牌进行加密。【75】（思想指导）
         4. 使用安全的加密算法，使用安全的协议（HTTPS、SSL/TLS)，更长的密钥长度，确保传递数据的完整性与安全性 。【65】（具体检查项）
         5. 使用加密协议确保基于cookie的身份验证的安全性。【66】（具体检查项）
         6. 使用IPSec（Ip协议的安全拓展）加密系统通信的整个过程，并且该过程不需要开发人员手动控制。【67】（具体检查项）
         7. 使用数字签名与加密减少在OAuth 2.0流程的各个阶段错误发送和交换令牌所带来的常见风险和漏洞。【68】（具体检查项）
         8. 采用API security中规定的基于标准的现代安全机制。提供REST JSON消息级安全，例如JWT (JSON Web Token)、JWS (JSON Web Signature)和JWE (JSON Web Encryption)。JWS是JWT的基础结构，提供了签名功能，用于验证数据的完整性和真实性。而JWE则提供了加密功能，可以保护数据的机密性。【71】（具体检查项）
         9. 将Oauth与JWT结合起来会有很大的性能改进，并且对具体加密算法选取上，HS256加密算法拥有更好的性能。【72】（具体检查项）
         10. 使用VPN或SSL/TLS加密防止API流量被窃取。【73】（具体检查项）
         11. 物联网中MQTT作为一种轻量级的数据传输协议被广泛使用，但其并未提供较强的安全配套功能，建议使用VPN或TLS加密MQTT服务器与节点间的通信。【74】（具体检查项）
      2. 其它传输保护措施
         1. 利用OSI参考模型中的不同层次的功能保护两个实体间的通信。【76】（思想指导）
         2. 对于Web服务链，web服务可能会跨域多个中间件服务，仅使用传输层提供的点对点保护是不安全的，应提供基于应用层的Web服务安全性保护策略。【77】（思想指导）
         3. 确保数据完整性不被破坏。使用私钥对SOAP消息进行签名，并防止重放攻击。【78】（具体检查项）
         4. 在请求中包含时间戳来防止重放攻击。确保请求是最新的，并防止攻击者重复利用之前的请求来执行未授权的操作。【79】（具体检查项）
   2. 输入验证与数据清理
      1. 不要相信用户，不应将验证延迟到客户端。服务端至少需要采用与客户端一样严格的输入验证，对来自API的用户输入进行与来自Web UI的数据采用一样的处理方式。如果仅在客户端进行输入验证，攻击者可以绕开客户端限制利用web api在没有完整授权的情况下获取数据或进行攻击。谨慎对待用户输入，对用户输入进行处理与验证。【80】（思想指导）
      2. 攻击者可能绕开客户端验证，采用非法的请求方式，因此服务端不能对客户进行假设，必需时刻对输入进行处理与验证。【81】（思想指导）
      3. 输入验证和清理技术可以防范恶意代码或SQL注入。【82】（思想指导）
      4. 输入验证
         1. 任何时候都需要检查请求内容的类型、大小和长度，确保应用程序后续操作的所有数据都是有效且合法的。识别和防止通过改变输入数据结构来绕过安全授权检查的攻击。【84】（思想指导）
         2. 验证用户的注册信息是否满足一定规范，避免恶意注册。【91】（思想指导）
         3. 验证用户和服务器之间发送的内容类型，确保数据的格式正确，预防不支持的内容类型导致的处理错误或安全漏洞。【93】（思想指导）
         4. 通过输入验证、特别字符的逃逸处理，以及使用参数化查询等技术来防止SQL注入和XSS攻击。【94】（思想指导）
         5. 避免非法的请求输入导致的服务端的5xx错误。【95】（思想指导）
         6. 使用内置的函数检查输入，而不是使用自定义的验证逻辑。【83】（具体检查项）
         7. 服务端需对上传文件的文件扩展名与文件内容类型进行严格验证，阻止用户上传与预期不符或恶意的文件。【85】（具体检查项）
         8. 服务端需严格限制上传文件大小的最大值，以防止拒绝服务攻击。 【86】（具体检查项）
         9. 服务端需限制上传文件大小的最小值，以防止拒绝服务攻击。【87】（具体检查项）
         10. 服务端限制上传文件的文件名及其扩展名的长度应小于255个字符，大部分文件系统对文件名与扩展类型的长度有限制，超出该数值可能导致出错或产生意料之外的行为。【88】（具体检查项）
         11. 服务端在进行实际的语义与语法验证前，将数据转化为标准格式，从而更容易地识别和过滤恶意代码或攻击量。【89】（具体检查项）
         12. 服务端需限制单次上传文件的数量，以避免Dos攻击。【90】（具体检查项）
         13. 对用户注册时输入的密码建立多个要求。强密码的最小长度为10个字符，应该包含小写字母、大写字母。 防止密码泄露及身份验证失效。【92】（具体检查项）
         14. 避免直接将传入数据与内部对象绑定。【96】（具体检查项）
      5. 输入清理与转义
         1. 在使用正则表达式对用户输入进行清理时，需要谨慎设计正则表达式，避免包含模糊不清的部分（意味着在匹配输入时可能需要超线性的输入评估和处理时间）。【99】（思想指导）
         2. 无论来源如何，将所有字符串类型的值视为不可信，并在运行时进行验证与转义，从而有效避免XEE攻击 。【100】（思想指导）
         3. 转义所有的用户输入，以避免SQL注入 。【101】（思想指导）
         4. 除非在允许的位置，否则不能插入不可信的数据，以避免XSS攻击。 【102】（思想指导）
         5. 服务端需从上传的文件名及其扩展名中删除所有控制字符和Unicode字符。此外,特殊字符如 “;”, “:”, “>”, “<”, “/” ,”\”, ”。"、" \* "、" % "、" $ "等也应该被丢弃，避免攻击者使用这些字符插入恶意代码或命令或引发其它安全漏洞 。【97】（具体检查项）
         6. 如果服务端不需要适配Unicode字符，建议仅接受字母、数字与.作为文件名与扩展名且不能为空。（正则表达式 【a-zA-Z0- 9】{1,200}\.【a-zA-Z0-9】{1,10}) 。避免攻击者使用这些字符插入恶意代码或命令或引发其它安全漏洞 。【98】（具体检查项）
         7. 在HTML元素内容中插入不受信任的数据之前进行HTML转义，以避免XSS攻击。【103】（具体检查项）
         8. 在HTML公共属性中插入不可信数据之前，对属性进行转义，以避免XSS攻击。【104】（具体检查项）
         9. 在JavaScript数据值中插入不可信数据之前进行JavaScript转义，以避免XSS攻击。【105】（具体检查项）
         10. 在将不可信的数据插入HTML样式属性值之前，进行CSS转义，以避免XSS攻击。【106】（具体检查项）
         11. 在HTML URL参数值中插入不可信数据之前进行URL转义，以避免XSS攻击。【107】（具体检查项）
         12. 在解析注册/登录输入时，清理javascript://、data://、CRLF字符等。上传功能应该清理用户提供的文件名及EXIF标签。清理所有用户输入或任何暴露给用户的输入参数来防止XSS。【108】（具体检查项）
         13. 确保API返回的数据在客户端被适当地清理或转义，以防XSS攻击。【109】（具体检查项）
   3. 数据管理
      1. 保护敏感信息
         1. HTTP体/头部的ID比URL中的更易受攻击，需要额外注意。【116】（思想指导）
         2. 避免在URL中传递敏感信息。【118】（思想指导）
         3. 避免通过公开的Swagger（一款RESTFUL接口的文档在线自动生成+功能测试功能软件）文件暴露敏感信息。【119】（思想指导）
         4. 用户管理模块易受攻击，需要重点关注，避免用户敏感信息泄露。【120】（思想指导）
         5. 数据库中仅保存重要信息，API返回的其余数据在运行时通过API抽象层发送，防止攻击者通过拦截的部分信息获取用户的完整数据。【110】（具体检查项）
         6. 数据库中不应存储密码明文，建议存储密码的散列版本 ，防止针对数据库的攻击获取密码。【111】（具体检查项）
         7. 不在设备上存储敏感数据，避免对敏感数据进行硬编码。使用安全的密钥管理机制，避免系统数据泄露或遭受攻击后造成的敏感数据泄露 。【112】（具体检查项）
         8. 对数据库存储的密码或其它重要信息进行单向加密，防止因SQL注入等产生的用户隐私信息泄露 。可以使用Bcrypt存储密码哈希。【113】（具体检查项）
         9. 加密MQTT（Message Queuing Telemetry Transport)节点及其代理中的缓存数据，防止攻击者从中获取敏感信息。【114】（具体检查项）
         10. 确保ID随机分配，防止攻击者通过预测ID来访问或篡改数据。【115】（具体检查项）
         11. 移除或修改可能泄露信息的HTTP头部（如X-Powered-By、Server、X-AspNet-Version 等）。【117】（具体检查项）
      2. 其它数据管理措施
4. 所有私有信息用ASCII码编码并存储在单个数据库表中，可部分缓解SQL注入攻击。【121】（具体检查项）
5. 在数据库中存储用户上传的文件而不是在文件目录中存储，防止用户直接从URL中获取文件。【122】（具体检查项）
   1. 数据备份和恢复
      1. 定期备份API数据和配置。测试恢复过程。【123】（思想指导）
6. 监控与响应
   1. 审计与监控
      1. 记录并保护日志
         1. 记录操作日志，对系统进行全面监控。【124】（思想指导）
         2. Laas平台至少需要提供关于访问、执行命令和网络流量的日志文件，以便在遭受安全攻击时，通过日志快速定位并处理 。【125】（思想指导）
         3. 服务端需记录所有文件上传失败与成功的记录，方便运维人员排查问题.同时保护日志记录不被篡改并防止代码注入，避免攻击者掩盖攻击行为。 【126】（具体检查项）
         4. 记录用户输入的错误信息与尝试输入的数据 ，帮助运维与安全监控人员识别攻击并防范。【127】（具体检查项）
         5. 不明文记录敏感数据的日志，日志的安全保护级别可能较低，在日志中记录敏感数据可能会造成敏感数据泄露 。【128】（具体检查项）
      2. 监控与警报
         1. 监控网络流量。【129】（思想指导）
         2. 使用在正常API流量数据集上训练的无监督学习的异常检测模型监测系统流量，以便与学习到的正常流量模式进行比较。每当请求偏离正常模式时，就会触发安全操作，以防止/警报系统可能受到的恶意攻击。【130】（思想指导）
         3. 利用深度学习方法持动态对比分析多个连续的业务请求数据，发现未授权访问、用户身份认证失败等业务安全隐患 。【131】（思想指导）
         4. 在CI/CD流程中确保有一个审计系统，确保所有变更都被跟踪和记录，有助于识别潜在的安全风险和不规范的改动。【132】（思想指导）
         5. 通过API中介改善可观测性和监控，确保能及时发现并响应安全事件和性能问题的关键。【133】（思想指导）
   2. 防止资源耗尽与滥用
      1. 避免资源耗尽与DOS攻击
         1. 监控API的负载并在过载情况下断开连接或增加服务资源。【134】（思想指导）
         2. 使用更严格的速率限制，断开流量过高请求的连接，保护API免受过度流量和恶意超载尝试，避免资源耗尽，防止Dos攻击。【135】（思想指导）
         3. 断开未授权请求的IP地址连接，从而有效避免Dos攻击。【136】（具体检查项）
         4. 限制MQTT（Message Queuing Telemetry Transport)消息传递速率并采用MQTT分布式代理的方式减轻服务端负载，以避免Dos攻击。【137】（具体检查项）
         5. 针对物联网对互操作性的要求，使用Oauth时，授权服务器需要具有较高的带宽以处理用户请求，同时限制每个用户可获得的授权码和用户身份令牌数量，避免针对授权过程的Dos攻击。【138】（具体检查项）
      2. 资源访问策略
         1. 已经被删除的资源不可再被访问，避免产生资源释放后使用错误。防止客户端绕过资源配额并破坏服务端状态。【139】（具体检查项）
         2. 未创建完成的资源必须无法访问，并且不会影响服务端状态（例如不会增加资源的数量，不会影响其它资源的命名）。【140】（具体检查项）
      3. 负载均衡
         1. 配置负载限制，确保服务器不会因超过处理能力的请求而崩溃，保持系统稳定。【141】（思想指导）
   3. 事故响应
      1. 日志记录和监控系统可以有效识别和处理安全事件。如在黑客攻击或数据泄露的情况下，检查以前的日志以查看数据访问情况，要求用户更改密码。【142】（思想指导）
      2. 开发并定期更新应对API违规事件的事故响应计划。制定应对文档差异和API偏移的应急计划。【143】（思想指导）
7. API开发与运维
   1. 开发中的API使用规范
      1. 加密API使用
         1. Java中，不应使用字符串对象存储密码。【144】（具体检查项）
         2. 加密API在密钥构造时应加入随机参数，密钥生成的迭代次数不应太低。 Java中，PBEKeySpec(PBE加密算法)构造函数的第二个参数是不安全的，其应该是随机的。第三个参数也是不安全的。它指定对密码进行散列以获得加密密钥的迭代次数。然而，这个迭代计数参数不应该小于1000。【145】（具体检查项）
         3. 使用加密API时，对相关功能前后调用的语句顺序以及选取的加密API都需要检查。【146】（具体检查项）
      2. 安全访问数据库
         1. 在后端仅通过参数化查询方式访问数据库，而不是手动创建。【147】（思想指导）
         2. 使用存储过程而不是手动拼接SQL语句从而避免SQL注入。【148】（具体检查项）
         3. 使用预处理语句（参数化查询）进行数据库操作从而避免SQL注入。 【149】（具体检查项）
         4. 开发人员可以使用安全查询对象和SQL DOM，从而实现安全可靠的数据库访问。【150】（具体检查项）
      3. 配置与依赖管理
         1. 对应用程序的配置进行统一的管理，不同应用的配置文件应被及时更新。【151】（思想指导）
         2. 验证依赖的安全性。定期更新API及其插件，移除未使用的依赖和组件，确定依赖没有使用过时的API或未上线的测试版本的API。【152】（具体检查项）
         3. 在选择新依赖时，只从官方和安全的链接添加，以防止引入含有恶意代码或安全漏洞的依赖。【153】（具体检查项）
         4. 应用部署前关闭调试模式，设置为生产模式。防止在生产环境中意外地暴露调试信息或未完成的功能，保护应用免受安全风险。【154】（具体检查项）
         5. 避免使用旧版本API，旧版本API可能包含已知的安全漏洞，缺乏最新的安全措施，更容易受到攻击。【155】（具体检查项）
         6. 确保所有重定向都是安全和合法的，开放的重定向可能使API容易受到网络钓鱼或恶意软件传播的攻击。【156】（具体检查项）
         7. 移动/网页应用API的依赖可能存在差异，其安全实现方式也可能不同。【157】（具体检查项）
      4. 其它
         1. 使用安全的数据类型与安全的创建方式。【158】（思想指导）
         2. 服务器到服务器之间进行的API调用不应从应用程序中进行。【161】（思想指导）
         3. 使用安全接收器，其与原始的DOM sink接受的字符串不同，只接受安全类型的值。【159】（具体检查项）
         4. 防止攻击者利用特定功能，比如.NET的Path.Combine来执行攻击。【160】（具体检查项）
         5. 对象模式中将不应修改的属性标记为readOnly。【162】（具体检查项）
         6. API使用CamelCase格式来查询参数和请求/响应体，保持API的一致性和可读性，符合现代编程实践。【163】（具体检查项）
   2. API调用与管理
      1. 为用户及调用方提供恰当的提醒
         1. 提供关键按键操作确认和两次认证，提醒或阻止可能导致安全问题的操作。【164】（具体检查项）
         2. 当调用方没有足够的权限时，被调用者应提供异常抛出功能以及适当的提醒，避免过度授权风险。【165】（具体检查项）
         3. 服务端需为用户提供上传文件失败的原因与提醒，避免用户反复上传文件引起服务器资源浪费。【166】（具体检查项）
      2. 其它
         1. 识别和禁止使用具有XSS攻击倾向的DOM处理API。【167】（思想指导）
         2. 企业应该定期进行API资产清点，以便更好地管理已有的API，同时识别并废除影子API与僵尸API，避免攻击者就会利用这两类本应不能访问的API入侵企业系统，造成损失。【168】（思想指导）
         3. 在社交网络中使用API时，数据应在客户端动态加载，而不是通过JSONP在服务器端静态加载，以避免CSRF攻击。【169】（思想指导）
         4. 浏览器厂商可以对屏幕共享API实施约束，增强浏览器安全性。【170】（思想指导）
         5. 网站可以通过实现X帧选项和验证ORIGIN头等控件来防御利用屏幕共享的基本攻击。【171】（具体检查项）
   3. API测试与评估
      1. 测试用例与代码
         1. 检查最不常用的功能，不常用的功能可能未经频繁审查或更新，因此更有可能含有安全漏洞。【178】（思想指导）
         2. 进行API代码静态和动态测试，可以在代码部署前发现潜在的安全问题，减少生产环境中的风险。【179】（思想指导）
         3. 使用NLP进行测试用例生成，NLP技术可以动态创建带有测试数据的测试用例，这些可用于揭露功能和非功能问题，包括安全缺陷）以及报告分析。【172】（具体检查项）
         4. 使用Burp Suite进行API端点管理(Burp Suite是一种常用的安全测试工具，可以帮助管理和测试API端点，确保它们的安全性和正确性)。使用工具如Traceable AI追踪数据并识别潜在测试案例。【173】（具体检查项）
         5. 通过发送大量请求值测试API的容错性。【174】（具体检查项）
         6. 使用有效输入和预期响应对每个API调用进行类单元测试。【175】（具体检查项）
         7. 测试API调用序列以验证预期的用例，确保API的整体功能和各个端点间的交互符合业务逻辑和用户需求。【176】（具体检查项）
         8. 使用无效请求进行负面测试，以评估API的错误处理和安全性。【177】（具体检查项）
         9. 渗透测试模拟攻击者的攻击手段，帮助识别和修复在实际部署前可能未被发现的安全漏洞。在类似生产环境的条件下对API端点进行渗透测试。【180】（具体检查项）
      2. 评估API安全性
         1. 编写安全代码（输入验证、异常处理避免硬编码）并进行代码质量检测。【181】（思想指导）
         2. 使用可以在API请求、响应、OAS和其他合规性数据源的数据集上进行训练的机器学习系统，确保应用程序满足所需的标准。【182】（思想指导）
         3. 使用自动化工具帮助开发人员，并帮助检测权限过高和配置不一致的代码，减少手动过程导致的错误与人力浪费。【183】（思想指导）
         4. 利用REST可预测性管理API端点(通过分析REST API的结构和命名约定来发现可能的管理端点，以便进行进一步的安全测试或管理)。【185】（具体检查项）
         5. 检查老版本的iOS和Android客户端的证书固定，老版本的客户端可能未启用证书固定，可能易受中间人攻击。【184】（具体检查项）
   4. API响应与数据管理
      1. 设置正确的响应格式
         1. 确保API的响应符合预定义的格式和标准，保证接口的一致性和可靠性。【189】（思想指导）
         2. API响应设置适当的Content-type。JSON格式使用“application/json; charset=utf-8”,xml格式使用“text/xml;charset=utf-8”，确保客户端对响应数据进行正确处理，避免解析响应时产生注入风险。【186】（具体检查项）
         3. 无论用户如何调用，所有的API响应都因具有一致的数据格式，而不是对无效调用返回HTML响应，避免针对返回HTML进行的污染攻击。【187】（具体检查项）
         4. 使用X-Frame-Options和Content-Security-Policy等HTTP响应头，防止攻击者利用点击劫持和内容嗅探来窃取用户信息或执行恶意操作。 【188】（具体检查项）
      2. 最小化响应内容
         1. 最小化响应内容以限制敏感信息的泄露。确保错误信息不会泄露敏感或系统内部的细节。【190】（具体检查项）
         2. 分页展示列表形式的可变数量的API，确保API在处理大量数据时的性能和安全性，防止因配置不当导致的拒绝服务攻击（DoS）。【191】（具体检查项）