Web 安全测试学习手册

1. 配置管理测试

1.1. 远程代码执行漏洞

1.2. Slow HTTP DOS（慢速拒绝服务）漏洞

1.3. 点击劫持：X-Frame-Options 头丢失漏洞

1.4. 服务器启用了不安全的 HTTP 方法漏洞

1.5. 中间件版本信息泄露漏洞

1.6. 服务器端目录遍历漏洞

1.7. 中间件解析漏洞

1.8. IIS 短文件名漏洞

1.9. 应用程序未容错漏洞

1.10. SVN 文件泄露漏洞

1.11. OpenSSL 心脏出血漏洞

1.12. SSL/TLS “受戒礼” 漏洞

1.13. SSL POODLE 漏洞

1.14. 分布式部署文件可读漏洞

1.15. SSI（Server-Side Includes Injection）漏洞

1.16. 未授权访问漏洞

1.17. GET 请求传输敏感信息

2. 数据验证测试

2.1. XSS(Cross Site Scripting)跨站脚本攻击漏洞

2.2. SQL 注入(SQL injection)漏洞

2.3. 任意文件读取漏洞

2.4. 任意文件上传漏洞

2.5. XXE(XML External Entity Injection)（XML 实体注入）漏洞

2.6. 代码注入漏洞

2.7. 命令注入漏洞

2.8. 任意文件下载漏洞

2.9. 本地文件包含漏洞

2.10. 远程文件包含漏洞

2.11. 框架注入漏洞

2.12. 链接注入漏洞

2.13. SSRF(Server-Side Request Forgery)服务器请求伪造漏洞

2.14. CSRF(Cross Site Request Forgery)跨站请求伪造漏洞

2.15. 任意 URL 跳转漏洞

2.16. JSON 劫持漏洞

2.17. JSONP(JSON with Padding)跨站劫持漏洞

2.18. CORS()Cross-Origin Resource Sharing 跨域资源读取漏洞

2.19. Flash 跨域劫持漏洞

2.20. SOME(Same Origin Method Execution)同源方法执行漏洞

2.21. XSSI (Cross Site Script Inclusion) 跨站脚本包含漏洞

2.22. SSTI(Server-Side Template Injection)服务端模板注入漏洞

2.23. Html 注入漏洞

2.24. Server-Side Includes (SSI) Injection 服务器端包含注入漏洞

2.25. Host Header Attack host 头攻击漏洞

5.2. 图形验证码绕过漏洞

5.3. 短信验证码绕过漏洞

5.4. 短信验证码重放漏洞

5.5. 业务流程绕过漏洞

5.6. 加密算法脆弱漏洞

5.7. 支付逻辑漏洞漏洞

5.8. 竞争条件(HTTP 并发)漏洞

5.9. 前端认证绕过漏洞

5.10. 验证码客户端回显漏洞

5.11. 业务流程绕过漏洞

5.12. 业务安全测试关键点

5.12.1. 技术篇

- 01.登录认证模块测试
    - 暴力破解测试（暴力破解漏洞）
        - 1.增加图形验证码，图形验证码采用服务器端校验，登录失败一次，验证码变换一次。

2.配置登录失败次数限制策略，如在同一用户尝试登录情况下，5 分钟内连续登录失败超过 6 次，则禁止用户在 3 小时内登录系统。
3.在条件允许的情况下，增加手机接收短信验证码或邮箱接收邮件验证码，实现双因素认证的防暴力破解机制。
4.敏感字段采用强加密传输，建议采用 md5($pass.$salt)或者 md5($salt.$pass)加密，不建议采用简单 MD5 加密甚至 base64 编码处理。

- 本地加密传输测试（明文传输漏洞）（未加密登录请求漏洞）

  - 1.在架设 Web 应用的服务器上部署有效的 SSL 证书服务，采用 https 协议加密传输数据。
  2.敏感字段采用强加密传输，建议采用 md5($pass.$salt)或者 md5($salt.$pass)加密，不建议采用简单 MD5 加密甚至 base64 编码处理。

- Session 测试

  - Session 会话固定测试

    - 在客户端登录系统时，应首先判断客户端是否提交浏览器的留存 Session 认证会话属性标识，客户端提交此信息到服务器时，应及时销毁浏览器留存的 Session 认证会话，并要求客户端浏览器重新生成 Session 认证会话属性标识。

  - Session 会话注销测试

    - 在用户注销或退出应用系统时，服务器应及时销毁 Session 认证会话信息并清空客户端浏览器 Session 认证会话属性标识。

  - Session 会话超时时间测试

- 对每个生成的 Session 认证会话配置生命周期（常规业务系统建议 30 分钟内），从而有效降低因用户会话认证时间过长而导致的信息泄漏风险。

- Cookie 仿冒测试

  - 建议对客户端标识的用户敏感信息数据，使用 Session 会话认证方式，避免被他人仿冒身份。

- 密文对比认证测试

  - 将密码认证过程及密文对比过程放置在服务器后台执行。发送用户名和密码到服务器后台，后台对用户提交的密码经过 MD5 算法加密后和数据库中存储的 MD5 密码值进行比对，如果加密值相同，则允许用户登录。

- 登录失败信息测试（用户名枚举漏洞）

  - 对系统登录失败提示语句表达内容进行统一的模糊描述处理，从而提高攻击者对登录系统用户名及密码的可猜测难度，如“登录账号或密码错误”、“系统登录失败”等。

- 02.业务办理模块测试

  - 订单 ID 篡改测试

    - 后台查看订单时可通过 Session 机制判断用户身份，做好平行权限控制，服务端需要校验相应订单是否和登录者的身份一致，如发现不一致则拒绝请求，防止平行权限绕过漏洞泄漏用户敏感个人隐私信息。

  - 手机号篡改测试

- 后台请求要通过 Session 机制判断用户身份，如需要客户端传输手机号码，则服务端需要校验手机号是否和登录者的身份一致，如发现不一致则拒绝请求，防止平行权限绕过。另外，对于手机 APP 程序，不要过于相信从手机中直接读取的手机号码，还是要做常规的身份认证，规范登录流程，防止未授权登录。

- 用户 ID 篡改测试

  - 后台功能请求要通过 Session 机制判断用户身份，不要相信客户端传来的用户 ID，如果确实需要客户端传输 userid，则服务端需要校验 userid 是否和登录者的 Session 身份一致，如发现不一致则拒绝请求，防止被攻击者篡改，未授权访问他人账号内容。

- 邮箱和用户篡改测试

  - 用户登录后写信、发送信息时要通过 Session 机制判断用户身份。如果需要客户端传输邮箱、发件人，服务端需要校验邮箱、发件人是否和登录者的身份一致，如发现不一致则拒绝请求，防止被攻击者篡改用于钓鱼攻击。

- 商品编号篡改测试

  - 建议商品金额不要在客户端传入，防止被篡改。如果确实需要在客户端传输金额，则服务端在收到请求后必须检查商品价格和交易金额是否一致，或对支付金额做签名校验，若不一致则阻止该交易。

- 竞争条件测试

  - 在处理订单、支付等关键业务时，使用悲观锁或乐观锁保证事务的 ACID 特性（原子性、一致性、隔离性、持久性），并避免数据脏读（一个事务读取了另一个事务未提交的数据），解决竞争条件和并发操作可能带来的相关业务问题。

- 03.授权访问模块测试

  - 未授权访问测试（未授权访问漏洞）

    - 未授权访问可以理解为需要安全配置或权限认证的地址、授权页面存在缺陷，导致其他用户可以直接访问，从而引发重要权限可被操作、数据库、网站目录等敏感信息泄漏，所以对未授权访问页面做 Session 认证，并对用户访问的每一个 URL 做身份鉴别，正确校验用户 ID 及 Token 等。

  - 越权测试

    - 平行越权测试

    - 垂直越权测试

    - 1.使用安全配置，对敏感服务接口使用白名单访问控制列表。

    2.验证一切来自客户端的参数，重点是和权限相关的参数，比如用户 ID 或者角色权限 ID 等。

    3.session ID 和认证的 token 做绑定，放在服务器的会话里，不发送给客户端。

    4.对于用户登录后涉及用户唯一信息的请求，每次都要验证检查所有权，敏感信息页面加随机数的参数，防止浏览器缓存内容。

    5.把程序分成匿名，授权和管理的区域，通过将角色和数据功能匹配。

    6.不使用参数来区分管理员和普通用户。

7.服务端需校验身份唯一性，自己的身份只能查看、修改、删除、添加自己的信息。

- 04.输入输出模块测试
  - SQL 注入测试
    - 数字型注入
    - 字符型注入
    - 搜索型注入
    - 1.使用参数检查的方式在参数拼接进 SQL 语句前进行过滤或者校验，拦截带有 SQL 语句的参数传入应用程序。
      2.对 SQL 语句的语义进行完整性检查，确认语义没有发生变化。
      3.使用预编译处理的方式处理拼接了用户可控参数的 SQL 语句。
      4.Web 应用程序接入数据库服务器使用的用户不应该为系统管理员，用户角色应遵循最小权限原则。
      5.定期审计数据库执行日志，查看是否存在应用程序正常逻辑之外的 SQL 语句执行痕迹。
  - XSS 测试
    - 反射型 XSS
    - 存储型 XSS
    - DOM 型 XSS
    - 1.将重要的 Cookie 标记为 http only，使 javascript 中的 document.cookie 语句不能获取到 Cookie。
      2.输入检查：

在构造白名单的过程中需要保证在不影响用户体验的同时，尽可能杜绝一切不必要的输入内容，只允许用户输入我们期望的数据。 例如：年龄的 textbox 中，只允许用户输入数字，而数字之外的字符都过滤掉。

3.输出检查：

对数据进行 html encode 处理，过滤或移除特殊的 html 标签。

例如：

<script>， <iframe> ，< for <， > for >， &quot for

过滤 javascript 事件的标签。

例如：

"onclick=", "onfocus" 等等

建议过滤关键字为：

[1] < 左尖括号

[2] > 右尖括号

[3] " 双引号

[4] ' 单引号

[5] ` 反引号

[6] % 百分号

[7] (左圆括号

[8]) 右圆括号

[9] ; 分号

[10] / 正斜杠

[11] \ 反斜杠

[12] [左中括号

[13]] 右中括号

[14] & 连接符号

[15] # 井号

比如把<编码为<。

4.其他参考：

富文本过滤库

ruby： https://github.com/rgrove/sanitize

php： https://github.com/ezyang/htmlpurifier

javascript： https://github.com/leizongmin/js-xss

https://github.com/cure53/DOMPurify

更多：

https://github.com/search?o=desc&q=xss&ref=searchresults&s=stars&type=Repositories&utf8=%E2%9C%93。

- 命令执行测试

  - 1.使用低权限用户执行应用程序。

  2.设置重要系统命令禁止低权限用户执行。

  3.配置或代码过滤危险函数。

  4.升级到修复后的组件版本。

- 05.回退模块测试

  - 回退测试

    - 对于业务流程有多步的情况，如修改密码或重置密码等业务，首先判断该步骤的请求是否是上一步骤的业务所引起的，如果不是则返回错误提示或页面失效。

- 06.验证码机制测试

  - 验证码暴力破解测试

    - 1.设置验证码的失效时间，建议为 180 秒。

    2.限制单位时间内验证码的失败尝试次数，如 5 分钟内连续失败 5 次即锁定该账号 15 分钟。

  - 验证码重复使用测试

- 在验证码第一次认证成功后，服务器端清空认证成功的 Session，这样就可以有效防止验证码一次认证成功后重复使用的问题。

- 验证码客户端回显测试

  - 1.禁止验证码在本地客户端生成，应采用服务器端验证码生成机制。
    2.设置验证码的时效性，如 180 秒过期。
    3.验证码应随机生成，且使用一次即失效。

- 验证码绕过测试

  - 在服务端增加验证码的认证机制，对客户端提交的验证码进行二次校验。

- 验证码自动识别测试

  - 1.增加背景元素的干扰，如背景色、背景字母等。
    2.字符的字体进行扭曲、粘连。
    3.使用公式、逻辑验证方法等作为验证码，如四则运算法、问答题等。
    4.图形验证码和使用者相关，比如选择联系人头像、选择购买过的物品等作为验证码。

- 07.业务数据安全测试

  - 商品支付金额篡改测试

    - 商品信息，如金额、折扣等原始数据的校验应来自于服务器端，不应接受客户端传递过来的值。

  - 商品订购数量篡改测试

- 服务端应考虑交易风险控制，对产生异常情况的交易行为（如用户积分数额为负值，兑换库存数量为 0 的商品等）应当直接予以限制、阻断，而非继续完成整个交易流程。

- 前端 JS 限制绕过测试

  - 商品信息，如金额、折扣、数量等原始数据的校验应来自于服务器端，不应该完全相信客户端传递过来的值。类似的跨平台支付业务，涉及平台之间接口调用，一定要做好对重要数据，如金额、商品数量等的完整校验，确保业务重要数据在平台间传输的一致。

- 请求重放测试

  - 用户每次订单 Token 不应该能重复提交，避免产生重放订购请求的情况。在服务器订单生成关键环节，应该对订单 Token 对应的订购信息内容、用户身份、用户可用积分等进行强校验。

- 业务上限测试

  - 在服务器端应该对订单 Token 对应的订购信息内容、用户身份、用户可用积分等进行强校验。服务端应考虑交易风险控制，对产生异常情况的交易行为（如用户积分数额为负值，兑换库存数量为 0 的商品等）应当直接予以限制、阻断，而非继续完成整个交易流程。

- 08.业务流程乱序测试

  - 业务流程绕过测试

    - 对敏感信息如身份 ID、账号密码、订单号、金额等进行加密处理，并在服务端对其进行二次对比。

- 09.密码找回模块测试

- 验证码客户端回显测试
    - 避免返回验证码到响应包中，验证码一定要放在服务端校验。
- 验证码暴力破解测试
    - 对用户输入的验证码校验采用错误次数限制并提高验证码的复杂度。
- 接口参数账号修改测试
    - 对找回密码的 Token 做一对一的校验，一个 Token 只能修改一个用户，同时一定要保证 Token 不泄漏。
- Response 状态值修改测试
    - 注意不要在前端利用服务端返回的值判断是否可以修改密码，要把整个校验环节交给服务端校验。
- Session 覆盖测试
    - Session 覆盖类似于账号参数的修改，只是以控制当前 Session 方式篡改了要重置密码的账号，在重置密码请求中一定要对修改的账号和凭证是否一致做进一步的校验。
- 弱 Token 设计缺陷测试
    - 密码找回的 Token 不能使用时间戳或者用户邮箱和较短有规律可循的数字字符，应当使用复杂的 Token 生成机制让攻击者无法推测处具体的值。
- 密码找回流程绕过测试

- 防止跳过验证步骤一定要在后端逻辑校验中确认上一步流程已经完成。

- 任意用户密码重置的姿势
  - 01.验证码不失效
    - 找回密码的时候获取验证码缺少时间限制，仅是判断验证码是否正确未判断验证码是否过期。
    - 通过枚举找到真正的验证码，输入验证码完成校验。
  - 02.验证码直接返回
    - 输入手机号后点击获取验证码，验证码在客户端生成，并直接返回在 Response 以方便对接下来的验证码进行对比。
    - 直接输入目标手机号码，点击获取验证码，并观察返回包即可。在返回包中得到目标手机号获取的验证码，进而完成验证，重置密码成功。
  - 03.验证码未绑定用户
    - 输入手机号和验证码进行重置密码的时候，仅对验证码是否正确做了判断，未对该验证码是否与手机号匹配做判断。
    - 在提交手机号和验证码的时候，替换手机号为他人手机号测试，成功通过验证并重置他人密码。
  - 04.修改接收的手机或邮箱
    - 用户名、手机号、验证码这三者没有统一进行验证，仅判断了三者中手机号和验证码是否匹配和正确，如果正确则判断成功进行下一流程。

- 输入用户名获取验证码，修改接收验证码的手机号为自己的号码，自己手机成功接收验证码，提交到网站进行验证，验证成功进入下一个流程。

- 05.本地验证码的绕过

  - 客户端在本地进行验证码是否正确的验证，而该判断结果也可以在本地修改，最终导致欺骗客户端，误以为我们已经输入了正确的验证码。

  - 重置目标用户，输入错误验证码，修改返回包，把错误改成正确，即可绕过验证步骤，最终重置用户密码。

- 06.跳过验证步骤

  - 对修改密码的步骤没有做校验，导致可以输入最终修改密码的网址，直接跳转到该页面，然后输入新密码达到重置密码的目的。

  - 首先使用自己的账号走一遍流程，获取每个步骤的页面链接，然后记录页面 3 对应的输入新密码的链接，重置他人密码时，获取验证码后，直接输入页面 3 链接到新密码的界面，输入密码即可重置成功。

- 07.未校验用户字段的值

  - 在整个重置密码流程中，只对验证码和手机号做了校验，未对后面设置新密码的用户身份做判断，导致在最后一步通过修改用户身份重置他人的密码。

  - 使用自己的手机号走流程，在走到最后一个设置密码的流程时，修改数据包里的用户信息。

- 08.修改密码处 id 可替换

- 修改密码的时候，没有对原密码进行判断，且根据 id 的值来修改用户的密码，类似的 SQL 语句：update user set password = 'qwer1234' where id = '1'修改数据包里的 id 的值，即可修改他人的密码。

- 修改自己用户密码，抓取数据包，替换数据包中用户对应的 id 值，即可修改他人的密码。

- 09.Cookie 值的替换

  - 重置密码走到最后一步的时候，仅判断唯一的用户标识 Cookie 是否存在，并没有判断该 Cookie 有没有通过之前重置密码过程的验证，导致可替换 Cookie 重置他人的密码。

  - 重置自己用户密码到达最后阶段，抓到数据包，并在第一阶段重新获取用户 Cookie，替换 Cookie 到我们抓到的数据包中，发包测试。

- 10.修改信息时替换字段值

  - 在执行修改信息的 SQL 语句的时候，用户的密码也当作字段执行了，而且是根据隐藏参数 loginid 来执行的，这样就导致修改隐藏参数 loginid 的值，就可以修改他人的用户密码。

  - 修改个人资料的时候，抓取数据包，然后来修改数据包中的参数和对应的值，参数名一般可以在其他地方找到，替换隐藏参数即可修改他人的密码等信息。

- 10.业务接口调用模块测试

  - 接口调用重放测试

- 1.对生成订单环节采用验证码机制，防止生成数据业务被恶意调用。

  2.每一个订单使用唯一的 Token，订单提交一次后，Token 失效。

- 接口调用遍历测试

  - 在 Session 中存储当前用户的凭证或者 id，只有传入凭证或者 id 参数值与中的一致才返回数据内容。

- 接口调用参数篡改测试

  - 1.在 Session 中存储重要的凭证，在忘记密码、重新发送验证码等业务中，从 Session 获取用户凭证而不是从客户端请求的参数中获取。

    2.从客户端处获取手机号、邮箱等账号信息，要与 Session 中的凭证进行对比，验证通过后才允许进行业务操作。

- 接口未授权访问/调用测试

  - 1.采用 Token 校验的方式，在 url 中添加一个 Token 参数，只有 Token 验证通过才返回接口数据且 Token 使用一次后失效。

    2.在接口被调用时，后段对会话状态进行验证，如果已经登录，便返回接口数据；如果未登录，则返回自定义的错误信息。

- Callback 自定义测试

  - 1.严格定义 HTTP 响应中的 Content-Type 为 json 数据格式

    Content-Type: application/json;charset=UTF-8

    2.建立 callback 函数白名单，如果传入的 callback 参数值不在白名单内，跳转到同意的异常界面阻止其继续输出。

    3.对 callback 参数和 json 数据输出进行 HTML 实体编码来过滤掉“<”、“>”等字符。

- WebService 测试

- 1.为 WebService 添加身份认证，认证成功后才允许访问和调用。

2.WebService 中接收输入参数的函数，在后端应该对输入参数进行过滤及净化，在处理后才能入库查询。

3.在敏感功能的函数中，添加密码认证，认证后才允许调用敏感功能的函数。

5.12.2. 实践篇

- 01.帐号安全测试
  - 账号密码直接暴露在互联网上
    - 某企业数据库配置信息泄漏
    - 数千名员工信息泄露
  - 无限制登录任意帐号
    - SQL 注入漏洞可绕过登录限制
    - APP 客户端可以劫持任意帐号
  - 电子邮件账号泄漏事件
    - 邮件帐号引发的信息泄漏
  - 中间人攻击
    - SSL 证书欺骗攻击
    - SSL 劫持
  - 撞库攻击

        - 子站存在撞库风险

- 02.密码找回安全案例

    - 密码找回凭证可被暴力破解

        - 任意密码修改

    - 密码找回凭证直接返回给客户端

        - 密码找回凭证暴露在链接中

        - 加密验证码字符串返回给客户端

        - 网页源码中隐藏着密保答案

        - 短信验证码返回给客户端

    - 密码重置链接存在弱 Token

        - 使用时间戳的 MD5 做为密码重置 Token

        - 使用服务器时间做为密码重置 Token

    - 密码重置凭证与用户账号关联不严

        - 使用短信验证码找回密码

        - 使用邮箱 Token 找回密码

    - 重新绑定用户手机或邮箱

        - 重新绑定用户手机

        - 重新绑定用户邮箱

- 服务端验证逻辑缺陷
    - 删除参数绕过验证
    - 邮箱地址可被操控
    - 身份验证步骤可被绕过
- 在本地验证服务端的返回信息
    - 修改返回包绕过验证
- 注册覆盖
    - 已存在用户可被重复注册
- Session 覆盖
    - 通过 Session 覆盖方式重置他人密码

- 03.越权访问安全案例
    - 平行越权
        - 系统用户可越权查看其他用户个人信息
        - 网站用户可越权查看或修改其他用户信息
        - 普通用户可越权查看其他用户信息
    - 垂直越权
        - 普通用户权限越权提升为系统权限
        - 后台可越权添加管理员帐号

- 低权限用户可越权修改超级管理员配置信息
- 通过修改用户对应菜单类别可提升权限

- 04.OAuth2.0 安全案例
  - CSRF 漏洞导致绑定劫持
    - OAuth2.0 提供了 state 参数用于 CSRF 认证服务器将接收到的 state 参数按原样返回给 redirect_uri，客户端收到该参数并验证与之前生成的值是否一致。
    - 1.使用 CSRF-Token：业界一致的做法是使用一个 CSRF-Token。CSRF 攻击之所以能够成功是因为攻击者可以伪造用户的请求，对此最好的防御手段就是让攻击者无法伪造这个请求。因此，我们可以在 Http 请求中（千万不要放在 Cookie 中）以参数的形式添加一个随机的 CSRF-Token，并在服务器端检查这个 CSRF-Token 是否正确，如不正确或不存在，则可以认为是不安全的请求，拒绝提供相关服务。
      注意：
      如果网站同时还存在 XSS 漏洞时，上述 CSRF-Token 的方法将可能失效，因为 XSS 可以模拟浏览器执行操作，攻击者通过 XSS 漏洞读取 CSRF-Token 值后，便可以构造出合法的用户请求了。所以在做好 CSRF 防护的同时，相应的安全防护也应做好。
      2.验证 Referer 字段：在通常情况下，访问一个安全受限页面的请求来自于同一个网站，Http 头部中的 Referer 字段记录了该 Http 请求的来源地址，如果 Referer 中的地址不是来源于本网站则可认为是不安全的请求，对于该请求应予以拒绝。 这种方法简单易行，对于现有的系统只需在加上一个检查 Referer 值的过滤器，无需改变当前系统的任何已有代码和逻辑。
      但是，这种方法存在一些问题需要考虑：首先，Referer 的值是由浏览器提供的，虽然 Http 协议上有明确的要求，但是每个浏览器对于

Referer 的具体实现可能有差别，并且不能保证浏览器自身没有安全漏洞，将安全性交给第三方（即浏览器）保证，从理论上来讲是不可靠的；其次，用户可能会出于保护隐私等原因禁止浏览器提供 Referer，这样的话正常的用户请求也可能因没有 Referer 信息被误判为不安全的请求，无法提供正常的使用。最后一点，如果网站同时还存在 XSS 漏洞时，上述方法将可能失效，因为 XSS 可以模拟浏览器执行操作，构造出合法的用户请求，这样 Referer 字段记录的依然是本网站的地址从而可以绕过 Referer 校验。所以在做好 CSRF 防护的同时，相应的安全防护也应做好。

3.添加验证码机制（图形验证码或者短信验证码、邮件验证码）：
在用户提交数据之前，让用户输入验证码，或者用户在进行关键操作时，让用户重新输入密码进行验证。

4.校验非标准 Http 头部 X-Requested-With: XMLHttpRequest，我们可以在服务器端检查这个非标准 Http 头部 X-Requested-With: XMLHttpRequest 是否存在，如不存在，则可以认为是不安全的请求，拒绝提供相关服务。如果网站同时还存在 XSS 漏洞时，上述方法将可能失效，因为 XSS 可以模拟浏览器执行操作，构造出合法的 AJAX 用户请求，从而绕过非标准 Http 头部 X-Requested-With: XMLHttpRequest 校验，所以在做好 CSRF 防护的同时，相应的安全防护也应做好。

5.设置自定义的 Http 头部字段，比如 CSRF: Token。我们可以在服务器端检查这个自定义的 Http 头部字段是否存在，如不存在，则可以认为是不安全的请求，拒绝提供相关服务。

- 某社区劫持授权

- 05.在线支付安全案例总结

  - 某快餐连锁官网订单金额篡改

    - 总金额

- 某网上商城订单数量篡改
    - 数量
- 某服务器供应商平台订单请求重放测试
    - 请求重放
- 某培训机构官网订单其他参数干扰测试
    - 运费

6. 身份认证测试

6.1. 账号弱口令漏洞

6.2. 登录错误消息凭证枚举漏洞

6.3. 空口令攻击漏洞

6.4. 垂直越权漏洞

6.5. 平行越权漏洞

6.6. 暴力破解漏洞

6.7. 明文传输漏洞

6.8. 未加密登录请求漏洞

6.9. 未授权访问漏洞