安全checklist

权限系统 (注册/注册/二次验证/密码重置)

- [ ] 任何地方都使用 HTTPS.
- [ ] 使用 `Bcrypt` 存储密码哈希 (没有使用盐的必要 - `Bcrypt` 干的就是这个事).
- [ ] `登出`之后销毁会话 ID .
- [ ] 密码重置后销毁所有活跃的会话.
- [ ] OAuth2 验证必须包含 `state` 参数.
- [ ] 登陆成功之后不能直接重定向到开放的路径（需要校验，否则容易存在钓鱼攻击）.
- [ ] 当解析用户注册/登陆的输入时，过滤 javascript://、 data:// 以及其他 CRLF 字符.
- [ ] 使用 secure/httpOnly cookies.
- [ ] 移动端使用 `OTP` 验证时，当调用 `generate OTP` 或者 `Resend OTP` API 时不能把 OTP（One Time Password） 直接返回。（一般是通过发送手机验证短信，邮箱随机 code 等方式，而不是直接 response）
- [ ] 限制单个用户 `Login`、`Verify OTP`、 `Resend OTP`、`generate OTP` 等 API 的调用次数，使用 Captcha 等手段防止暴力破解.
- [ ] 检查邮件或短信里的重置密码的 token，确保随机性（无法猜测）
- [ ] 给重置密码的 token 设置过期时间.
- [ ] 重置密码成功后，将重置使用的 token 失效.

用户数据和权限校验

- [ ] 诸如`我的购物车`、`我的浏览历史`之类的资源访问，必须检查当前登录的用户是否有这些资源的访问权限.
- [ ] 避免资源 ID 被连续遍历访问，使用 `/me/orders` 代替 `/user/37153/orders` 以防你忘了检查权限，导致数据泄露。
- [ ] `修改邮箱/手机号码`功能必须首先确认用户已经验证过邮箱/手机是他自己的。
- [ ] 任何上传功能应该过滤用户上传的文件名，另外，为了普适性的原因（而不是安全问题），上传的东西应该存放到例如 S3 之类的云存储上面(用 lambda 处理)，而不是存储在自己的服务器，防止代码执行。
- [ ] `个人头像上传` 功能应该过滤所有的 `EXIF` 标签，即便没有这个需求.
- [ ] 用户 ID 或者其他的 ID，应该使用 RFC compliant 的 `UUID` 而不是整数. 你可以从 github 找到你所用的语言的实现.
- [ ] JWT（JSON Web Token）很棒.当你需要构建一个 单页应用/API 时使用.

安卓和 iOS APP

- [ ] 支付网关的 `盐（salt）` 不应该被硬编码
- [ ] 来自第三方的 `secret` 和 `auth token` 不应该被硬编码
- [ ] 在服务器之间调用的 API 不应该在 app 里面调用
- [ ] 在安卓系统下，要小心评估所有申请的 权限
- [ ] 在 iOS 系统下，使用系统的钥匙串来存储敏感信息（权限 token、api key、 等等） **不要** 把这类信息存储在用户配置里面

- [ ] 强烈推荐证书绑定（Certificate pinning）

安全头信息和配置

- [ ] 添加 CSP 头信息，减缓 XSS 和数据注入攻击. 这很重要.
- [ ] 添加 CSRF 头信息防止跨站请求伪造（CSRF）攻击.同时 添加 SameSite 属性到 cookie 里面.
- [ ] 添加 HSTS 头信息防止 SSL stripping 攻击.
- [ ] 添加 你的域名到 HSTS 预加载列表
- [ ] 添加 X-Frame-Options 防止点击劫持.
- [ ] 添加 X-XSS-Protection 缓解 XSS 攻击.
- [ ] 更新 DNS 记录，增加 SPF 记录防止垃圾邮件和钓鱼攻击.
- [ ] 如果你的 Javascript 托管在第三方的 CDN 上面，需要 添加 内部资源集成检查 。为了更加安全，添加require-sri-for CSP-directive 就不会加载到没有 SRI 的资源
- [ ] 使用随机的 CSRF token，业务逻辑 API 可以暴露为 POST 请求。不要把 CSRF token 通过 http 接口暴露出来，比如第一次请求更新的时候
- [ ] 在 get 请求参数里面，不要使用临界数据和 token。 暴露服务器日志的同时也会暴露用户数据

过滤输入

- [ ] 所有暴露给用户的参数输入都应该 过滤 防止 XSS 攻击.
- [ ] 使用参数化的查询防止 SQL 注入.
- [ ] 过滤所有具有功能性的用户输入，比如 csv导入
- [ ] 过滤一些特殊的用户输入，例如将 robots.txt 作为用户名，而你刚好提供了 coolcorp.io/username 之类的 url 来提供用户信息访问页面。（此时变成 coolcorp.io/robots.txt，可能无法正常工作）
- [ ] 不要自己手动拼装 JSON 字符串，不管这个对象有多么小。请使用你所用的语言相应的库或者框架来编写
- [ ] 过滤 那些有点像 URL 的输入，防止 SSRF 攻击
- [ ] 在输出显示给用户之前，过滤 输出信息

操作

- [ ] 如果你的业务很小或者你缺乏经验，可以评估一下使用 AWS 或者一个 PaaS 平台来运行代码
- [ ] 在云上使用正规的脚本创建虚拟机
- [ ] 检查所有机器没有必要开放的 端口
- [ ] 检查数据库是否没有设置密码或者使用默认密码，特别是 MongoDB 和 Redis
- [ ] 使用 SSH 登录你的机器，不要使用密码，而是通过 SSH key 验证来登录
- [ ] 及时更新系统，防止出现 0day 漏洞，比如 Heartbleed、Shellshock 等
- [ ] 修改服务器配置，HTTPS 使用 TLS1.2，禁用其他的模式。(值得这么做)
- [ ] 不要在线上开启 DEBUG 模式，有些框架，DEBUG 模式会开启很多权限以及后门，或者是暴露一些敏感数据到错误栈信息里面
- [ ] 对坏人和 DDOS 攻击要有所准备，使用那些提供 DDOS 清洗的主机服务
- [ ] 监控你的系统，同时记录到日志里面 (例如使用 New Relic 或者其他).
- [ ] 如果是 2B 的业务，坚持顺从需求。如果使用 AWS S3,可以考虑使用 数据加密 功能. 如果使用 AWS EC2，考虑使用磁盘加密功能（现在系统启动盘也能加密了）

关于人

- [ ] 开一个邮件组（例如：security@coolcorp.io）和搜集页面，方便安全研究人员提交漏洞
- [ ] 取决于你的业务，限制用户数据库的访问
- [ ] 对报告 bug、漏洞的人有礼貌
- [ ] 把你的代码给那些有安全编码观念的同伴进行 review (More eyes)
- [ ] 被黑或者数据泄露时，检查数据访问前的日志，通知用户更改密码。你可能需要外部的机构来帮助审计
- [ ] 使用 Netflix Scumblr 及时了解你的组织（公司）在社交网络或者搜索引擎上的一些讨论信息，比如黑客攻击、漏洞等等