34 | 账号密码泄露成灾，应该怎样预防?

2019-05-18 宝玉 来自北京

《软件工程之美》

你好，我是宝玉。我们日常总能看到各种与黑客和网络安全相关的新闻，而这其中大部分安全问题都和软件程序有关系。比如说像 CSDN 数据库泄露事件、携程泄露用户银行卡信息事件、有些电商网站用户可以篡改支付购买金额等等。

在软件项目开发时，安全是一个很容易被忽略的问题，但又可能会造成严重损失。所以我们在软件开发时有必要对安全问题引起重视，防患未然，构建安全软件。

今天，我将带你了解一下软件开发中的安全问题，学习如何构建安全的软件，以及出现了安全问题之后该怎么办。

安全问题本质是技术风险

如果你还记得《15 | 风险管理：不能盲目乐观，凡事都应该有 B 计划》这篇文章中的内容，我在其中提到，风险是指不确定的事件，一旦发生，将会造成消极的影响。

安全问题，本质上也是一种技术风险，没发生问题的时候一切都好，一旦发生就会有严重的影响。在对安全问题的应对上，你也可以借鉴对风险管理的方法来改进软件的安全问题，也就是风险识别、风险量化、应对计划和风险监控。

在做风险管理时，首先要做的就是识别风险和对风险量化，对于安全问题，你也可以先思考一下：软件项目中安全问题的主要来源是什么？搞清楚安全问题的来源，以及造成的后果，你就可以对软件中导致安全问题的情况有一个基本的识别和量化。

软件中的安全问题来源主要可以分为以下三大类。

第一类：恶意输入

很多我们熟知的软件安全问题都属于此类型，就是黑客通过恶意输入，然后绕过软件限制对系统进行攻击和破坏。

像 SQL 注入，就是黑客把 SQL 命令输入到软件的输入框或网页的 URL 查询参数，欺骗服务器，执行恶意的 SQL 命令。这样可以绕过密码验证，登录管理员账号，或者删除数据库数据，甚至控制服务器。

还有像 XSS 攻击，将恶意代码通过外部参数或者用户输入的方式植入网页中，获取用户的 Cookie 等敏感信息、盗用管理员权限，甚至非法转账。

这类问题都是由于恶意输入导致的，应对恶意输入的问题，最简单有效的方式就是对用户输入的数据，做严格的校验，格式化。

第二类：假冒身份

很多程序对于用户身份的校验比较弱，可能会导致黑客假冒用户身份做出超越权限的事情。

比如说，我见过有的网站，把后台入口隐藏起来，而没有做权限控制，导致黑客猜到地址后就可以进入后台操作。

还有的游戏后台不做验证，直接接收传入的数据，导致可以伪造游戏用户发送数据，破坏游戏平衡。

如果对用户的身份不做严格的验证，很可能就会导致假冒身份的安全问题。应对策略就是要对用户的身份做验证，尤其是涉及敏感权限的操作，甚至要做两重验证。

第三类：数据泄露

很多软件都储存了用户的敏感信息，比如用户帐号密码信用卡记录，或者服务器的敏感信息，比如数据库连接字符串、登录帐号密码，而这些数据是有被泄露风险的。

一些软件会把服务器上的敏感信息打包在程序中，而程序可能会被反编译导致敏感数据泄露。携程泄露用户银行卡信息事件中，就是因为把用户信用卡信息记录在日志中，日志泄露导致用户信用卡信息也被泄露，造成盗刷等严重问题。

还有 CSDN，对用户密码明文存储到数据库中，数据库泄露后，用户密码也跟着一起泄露了，而大多数用户习惯于在不同的网站也用相同的密码，导致在其他网站的密码也一起泄露了。

对于软件来说，**我们不能假设数据存储是安全的，而是要考虑到数据是有泄露的可能，提前做好预防措施，对敏感数据进行加密。**

在了解了这些常见的安全问题来源和可能带来的后果之后，我们在软件开发时，就可以对薄弱环节、重点问题进行提前预防，在开发时就考虑到可能的安全漏洞，做出科学的应对方案。

如何预防软件中的安全问题?

预防软件中的安全问题，也可以参考对风险管理的策略。在风险管理中，对风险识别和量化后，接下来就是要制定应对计划了。

很多开发人员觉得安全问题，只要在软件开发完成之后，测试阶段做一个安全测试就可以了，但这样做等于把安全问题留到了最后环节，是很难达到对安全问题进行高质量管控的。

一方面，对于安全测试来说，很难覆盖到所有可能存在的场景，可能会出现疏漏，导致安全漏洞被利用。另一方面，如果测试阶段发现安全问题，可能需要修改大量代码，甚至于要重新设计，这时候成本就太高了。

所以应对安全问题，最好的方式就是在整个生命周期中都做到重视安全问题，各个阶段都考虑到安全方面的问题，才能真正做到防患于未然，构建出安全的软件。

那么在软件开发的各个阶段，都需要如何考虑好安全方面的问题呢？

需求阶段

需求是软件项目的源头，**在确定需求，做产品设计的时候，不仅要考虑到功能上的需求，还要同时考虑到安全方面的要求。**这样当你在架构设计的时候，就不会轻易遗漏安全方面的架构设计。

尤其是对于一些外包项目，如果在需求中没有提出安全需求，大概率外包公司是不会帮你考虑这些需求的。

需求阶段，涉及用户输入的内容，需要考虑到可能的恶意输入，做出针对性预防措施；对于涉及用户权限的，要求有身份的验证，一些对安全要求极高的，可以在需求上就要求做双重验证；对于有敏感数据的，可以在需求上就要求对数据进行加密。

举个简单例子，比如说用户登录功能，如果让你提出安全方面的需求，你会考虑到哪些需求？这里我简单列几个功能参考：

登录网页使用 Https 或者在传输密码时加密；

增加图形校验码，避免恶意攻击；

密码失败次数过多，应该锁定用户一段时间；

记录用户登录 IP。

当你在需求阶段就提出了安全性的需求，设计、实现和测试时自然不会遗漏掉安全方面的内容，从源头上就让大家有了安全方面的意识。

设计阶段

在做设计架构时，最重要的事就是要把安全加入到设计目标，有了安全方面的设计目标，自然能找到很多安全相关的解决方案。

为了保障在设计时就考虑好安全方面的问题，在做架构设计方案的评审时，也需要增加安全方面的评审，确保有安全方面的考虑，确保技术方案切实可行。

现在架构设计领域，也有了一些业界公认的好的安全相关的设计原则，比如说攻击面最小化、权限最小化、纵深防御等。

攻击面最小化

攻击面就是指程序被用户直接访问到的部分，比如 API、网站等，这些暴露给用户的地方也是最可能被黑客攻击的地方。

暴露的面越多则风险越高，攻击面最小化的设计原则，就是说尽量减少暴露黑客可能发现并试图利用的攻击面数量。

举例来说，你的数据库应该关闭外网访问，避免黑客直接攻击数据库导致数据泄露。还有像对于一些复杂的多网站业务系统，实行单点认证，就可以让所有业务都在一个地方登录，你可以在这一个地方做到足够安全，这样所有网站的登录都是相对安全的。

权限最小化

权限最小化的设计原则就是对于系统的用户、文件访问、进程运行等，都只给予其能拥有的最小权限，这样可以保证一个应用程序或者网站被攻击、破解，能将损害降到最低。

举例来说，以前在部署 Asp.Net 程序的时候，运行 Asp.Net 的程序是单独的一个用户，这个用户所拥有的权限是只能运行程序所在目录，不能超出其目录范围，这样即使用户上传了恶意木马文件，那么也只能控制这一个目录，避免了进一步的损失。

纵深防御

纵深防御的设计原则，指的是从不同的维度去实施安全保护措施，从而缓解被攻击的风险。纵深防御并不是同一个安全方案要做两遍或多遍，而是要从不同的层面、不同的角度对系统做出整体的解决方案。

这里我给你举一个电商网站的例子（摘自：《代码未写，漏洞已出—架构和设计的安全》）。

> 国内中小电商，一半以上在早年都犯过这个错误，现在基本都修复了。电商的交易和支付系统之间流程是这样，一个人过来说老板我要买一台冰箱，多少钱？两千。OK，你把钱付给支付系统。因为支付请求也是在用户侧的浏览器里提交的。这个过程对用户是不可见的，但攻击者实际上可以修改这个数据。攻击者可以修改浏览器提交的数据，本来交易系统让他提交 2000 元，攻击者改为提交 1 元，然后支付系统就返回 OK，说我收到钱了。这个 OK 到交易系统那里，交易系统一看支付成功了，那就安排发货，1 元钱就把冰箱买到了。

你看，单独从支付系统和交易系统来看，设计上都没有问题，都对数据输入做了校验，但问题是没有站在一个系统的整体角度去考虑，没有考虑到不仅要校验交易有没有成功，还要校验交易的金额是不是匹配。

当然解决方案其实也很简单：

> 不要只反馈是否 OK，同时也把支付的金额和 OK 一起返回过去。是支付 2000 元 OK 还是 1 元 OK。这就解决了问题，现在的电商都改成这个设计了。

通过这样一些好的安全设计原则，就可以在设计阶段把很多安全问题预防好。

开发阶段

只是设计阶段做好安全相关的设计还不好，很多安全问题其实都是编码阶段时，没有好的编码习惯、没有良好的安全意识导致的。

对于开发阶段的安全问题预防，需要从以下几个方面入手。

编码规范中加入安全相关内容

对于用户输入的数据，需要有校验，防止恶意输入；对于涉及权限的操作，要检查用户权限；对于敏感数据要进行加密处理；对于用户的操作，要有日志记录；不能在日志中记录敏感信息等等。

要有代码审查

代码审查其实在我们专栏提过很多次，这也是预防安全问题一个行之有效的手段，通过代码审查，及时发现代码中的安全问题。

增加安全相关的自动化测试

现在有些安全工具，可以帮助对代码做安全检查，甚至可以和 CI 集成，如果增加相应的自动化安全测试代码，也可以第一时间对代码中的安全问题进行反馈。

测试阶段

在测试阶段，除了功能测试以外，增加对安全性方面的测试。除了增加相应的测试用例，也可以借助一些安全测试工具（可参考上一篇“软件测试工具”这篇文章）来进行测试。

上线维护

上线部署时，不部署源代码，只对编译后程序部署；删除 Debug 文件。

对服务器进行安全设置，比如说严格限制端口，只保留必须的端口；只对少数服务器开发外放服务；开启操作日志；对访问目录设置最小的权限。

通过对整个软件生命周期都做好安全方面的考虑，并落实到位，才能真正保证好软件的安全。

如果真的出现安全问题怎么办？

是不是我们在整个软件生命周期都做好安全方面的考虑，也落实到位，就完全不会有安全问题了？

安全问题就像程序的 Bug 一样，没有谁能保证绝对的安全，就像风险管理的最后一步风险监控那样，我们必须做好 Plan B，万一出现安全问题，马上应对，将损失降到最低。

首先，要设立应急的流程。当出现安全问题了，根据流程，知道该找谁，应该怎么去第一时间恢复生产，避免进一步损失。

其次，要分析程序的漏洞在哪里。通过分析日志，找出漏洞在哪里，才能针对性去修补漏洞。

最后，要总结原因。从错误中吸取教训，看问题是在哪个环节导致的，必要的话，就改进开发流程，避免类似的安全问题再次发生。

总结

今天我带你一起学习了如何构建安全的软件，软件的安全问题本质也是一种技术风险，可以借用风险管理的知识来帮助构建高安全性的软件。

软件安全问题主要来源是：恶意输入、假冒身份和数据泄露，要注意对输入数据的校验、对用户身份的校验和对敏感数据的加密。

构建高安全性软件，最好的方式就是在整个生命周期中都做到重视安全问题，各个阶段都考虑到安全方面的问题，才能真正做到防患于未然，构建出安全的软件。

安全问题就像程序的 Bug 一样，不能绝对避免，同时也要做好应对措施，在出现安全问题后，将损失降到最低，并且避免以后发生类似问题。

课后思考

通过对这篇文章的学习，你觉得你所在项目中，在安全方面有哪些可以改进的地方？你有哪些提升安全性的经验可以和大家一起分享的？欢迎在留言区与我分享讨论。

感谢阅读，如果你觉得这篇文章对你有一些启发，也欢迎把它分享给你的朋友。

精选留言 (8)

一步

2019-05-18

老师问一个有关code review的问题，code review是指把代码放到大屏幕上大家一起看呢？还是类似github上的合并代码的时候发个pr，然后另一个人对需要合并代码进行检查呢？检查通过后才同意合并请求

> 作者回复: 我觉得两种都算Code Review，只是形式不一样。
>
> 通常PR这种Code Review应该是贯穿到日常开发过程中的，每个人都可以去Review，有1-2个人Review通过就可以。合并的话不止是Code Review通过，还需要自动测试通过。
>
> 而大屏幕这种参与人多，成本高，属于偶尔针对特殊故障分析、学习研讨等目的才做的。

7

J.M.Liu

2019-05-22

订单和支付系统的那个例子里，支付系统的结果是返回给客户端，然后由客户端自己校对结果和订单或者把返回结果传给订单服务器校对吗？
这样还是很不安全啊。为什么不是把支付结果返回给服务端呢？

> 作者回复: 支付系统通过客户端返回订单系统，订单系统收到后需要再次根据订单跟支付系统确认，之前的漏洞就是出现在这个确认环节，只确认了是否收到，没有确认金额对不对。

3

alva_xu

2019-05-20

据我所知，安全测试也分白盒测试和黑盒测试两种，黑盒测试可以用fortify或appscan 来查，白盒测试可以通过code review来完成。在代码方面，写许多规范，即使写得很全，如果用手工测试，也很难确定开发人员是否真的按规范来写代码。所以，一般使用code review的工具来做。老师有没有好的安全代码检查的工具推荐?

作者回复: 我印象中Fortify也是可以做静态代码检查的。

除了Fortify和AppScan，我知道的还有Veracode、Checkmarx和CodeSecure。

但我没有实际用过，无法推荐哪个更合适。

共 2 条评论 >

3

纯洁的憎恶

2019-05-18

与风险管理一样，安全管理也需要从整个工程建设过程中整体考量。站在业务管理者角度，提高系统安全性的措施，可以在业务需求方面尽量避免安全负责度高的方案，在参与技术选型时尽可能兼顾安全要求。

作者回复: 👍谢谢总结分享

3

rocedu

2019-05-23

结合安全开发生命周期（SDL）就更好了。

作者回复: 👍谢谢娄老师补充。

1

ifelse

2022-07-05

我们不能假设数据存储是安全的，而是要考虑到数据是有泄露的可能，提前做好预防措施，对敏感数据进行加密。-"记下来"

calvins

2020-04-07

现在支付都有签名检验，后台金额检验，超时检验，在加https安全很多。

作者回复: 👍https现在很普及了

小老鼠

2019-10-02

所有的Bug都是风险，属于技术风险范畴。

作者回复: 👍