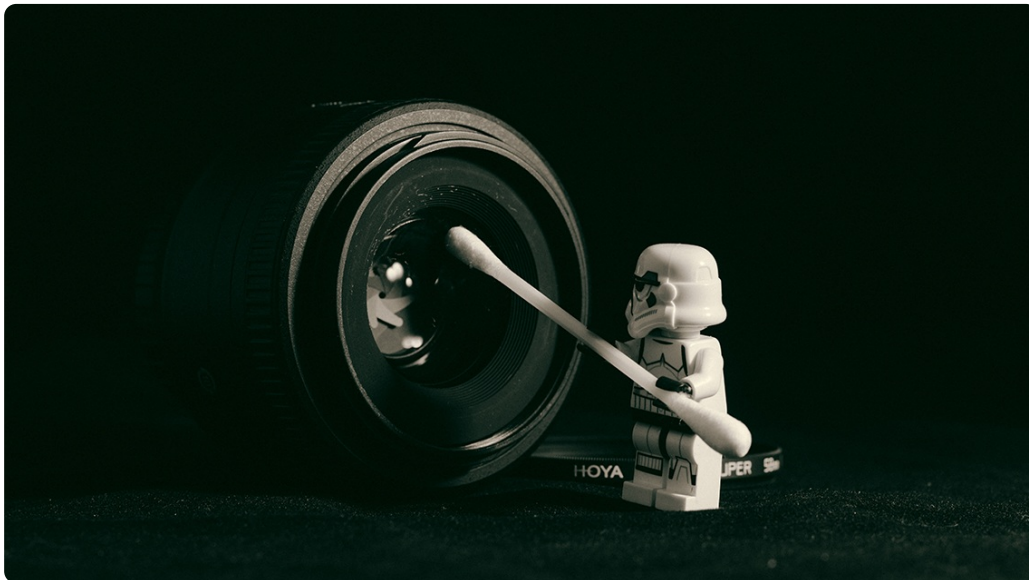


## 43 | 编写安全代码的最佳实践清单

范学雷 2019-04-12



00:00

讲述：刘飞 大小：16.44M

08:58

像以前一样，当大家看到“最佳实践清单”这个标题的时候，就意味着这一个模块又到了总结的时候了。

这一模块我们从代码安全的角度出发，探讨了如何编写安全的代码。首先我们再来重温一下，为什么需要安全的代码呢？

### 为什么需要安全的代码？

#### 1. 代码质量是信息安全的基础

大部分的信息安全事故，是由软件代码的安全缺陷引起的。没有安全质量保证的代码，建立不起有效、可信的信息系统。信息系统的安全，主要依赖的不是信息安全技术专家，而是我们每一个编写代码的工程师。

#### 1. 安全漏洞的破坏性难以预料

直到真实的安全问题发生之前，我们都难以预料软件的安全漏洞到底有多大的破坏性。一个小小的安全漏洞，如果被攻击者抓住了时机，就可以瞬间摧毁多年的苦心经营和良好声誉，把公司推到舆论的风口浪尖，甚至使公司面临毁灭性的风险和挑战。

#### 1. 安全编码的规则可以学得到

由于安全攻击技术的快速发展，安全编码涉及到的细节纷繁复杂，安全问题的解决甚至需要大规模、大范围的协作。编写安全的代码不是一件轻而易举的事情。但是，安全编码的规则和经验，

却是可以学习和积累的。**使用必要的安全管理工具，开展代码评审和交流，也可以加速我们的学习和积累，减少编写代码的安全漏洞。**

要想掌握安全编码的技术，熟练修复软件漏洞的实践，我们需要跨过意识、知晓、看到三道关卡。面对最新的攻击技术和安全问题，通过每一道关卡都障碍重重。我们要主动地跟踪安全问题的最新进展，学习最新的安全防护技术。

及时更新自己的知识，掌握难以学习到的知识和技能，也是构建和保持我们竞争力的一个重要办法。

## 编写安全代码的基本原则

### 1. 清楚调用接口的行为

**使用不恰当的接口，是代码安全风险的主要来源之一。**我们一定要了解、掌握每一个调用接口的行为规范，然后在接口规范许可的范围内使用它们。不要去猜测接口的行为方式，没有明文规定的行为，都是不可靠、不可信的行为。

### 2. 跨界的数据不可信任

跨界的数据面临两大问题：**一个是数据发送是否可信？另一个问题是数据传递过程是否可靠？**这两个有任何一个问题不能解决，跨界的数据都可能被攻击者利用。因此使用跨界的数据之前，要进行校验。

### 3. 最小授权的原则

信息和资源，**尤其是敏感数据，需经授权，方可使用。**所授予的权力，能够让应用程序完成对应的任务就行，不要授予多余的权力。

### 4. 减小安全攻击面

减小、简化公开接口，缩小可以被攻击者利用的攻击界面。比如，**设计更简单直观的公开接口，使用加密的数据传输通道，只对授权用户开放服务等等，这些措施，都可以减少安全攻击面。**

### 5. 深度防御的原则

使用纵深防御体系防范安全威胁。要提供深度的防御能力，不能仅仅依靠边界的安全。编写代码，**要采用谨慎保守的原则，要解决疑似可能出现的安全问题，要校验来源不确定的数据，要记录不规范的行为，要提供安全的应急预案。**

## 安全代码的检查清单

### 安全管理

- 有没有安全更新的策略和落实计划？
- 有没有安全漏洞的保密共识和规范？
- 有没有安全缺陷的评估和管理办法？
- 软件是不是使用最新的安全修复版？
- 有没有定义、归类和保护敏感信息？

- 有没有部署多层次的安全防御体系？
- 安全防御能不能运转良好、及时反应？
- 不同的安全防御机制能不能独立运转？
- 系统管理、运营人员的授权是否恰当？
- 有没有风险管理的预案和长短期措施？

## 代码评审

- 数值运算会不会溢出？
- 有没有检查数值的合理范围？
- 类、接口的设计，能不能不使用可变量？
- 一个类支持的是深拷贝还是浅拷贝？
- 一个接口的实现，有没有拷贝可变的传入参数？
- 一个接口的实现，可变的返回值有没有竞态危害？
- 接口的使用有没有严格遵守接口规范？
- 哪些信息是敏感信息？
- 谁有权限获取相应的敏感信息？
- 有没有定义敏感信息的授权方案？
- 授予的权限还能不能更少？
- 特权代码能不能更短小、更简单？
- 异常信息里有没有敏感信息？
- 应用日志里有没有敏感信息？
- 对象序列化有没有排除敏感信息？
- 高度敏感信息的存储有没有特殊处理？
- 敏感信息的使用有没有及时清零？
- 一个类，有没有真实的可扩展需求，能不能使用 final 修饰符？
- 一个变量，能不能对象构造时就完成赋值，能不能使用 final 修饰符？
- 一个方法，子类有没有重写的必要性，能不能使用 final 修饰符？
- 一个集合形式的变量，是不是可以使用不可修改的集合？
- 一个方法的返回值，能不能使用不可修改的变量？
- 类、方法、变量能不能使用 private 修饰符？
- 类库有没有使用模块化技术？
- 模块设计能不能分割内部实现和外部接口？
- 有没有定义清楚内部数据、外部数据的边界？
- 外部数据，有没有尽早地完成校验？
- 有没有标示清楚外部数据的校验点？
- 能不能跟踪未校验外部数据的传送路径？
- 有没有遗漏的未校验外部数据？
- 公开接口的输入，有没有考虑数据的有效性？

- 公开接口的可变化输出，接口内部行为有没有影响？
- 有没有完成无法识别来源的数据的校验？
- 能不能不使用序列化技术？
- 序列化的使用场景，有没有足够的安全保障？
- 软件还存在什么样风险？
- 有没有记录潜在的风险问题？
- 有没有消除潜在风险的长期预案？
- 有没有消除潜在风险的短期措施？
- 潜在的风险问题如果出现，能不能快速地诊断、定位、修复？

## 小结

学会编写安全的代码，是一个优秀的、专业的软件工程师的核心竞争力之一。与规范、经济的代码相比，安全的代码有很多不同的特点。

代码不规范和效率不高，业务也可以运转，然后慢慢优化，逐渐演进。但代码一旦出现安全问题，遭受攻击，损失立即就会反映出来，而且破坏性极大。

代码不规范，看的人立刻就会觉得很难受。代码的效率不高，业务运转不畅通，同样会有及时的反馈。就代码的安全层面来说，一般情况下直到攻击发生之前，我们可能都不知道代码是否存在安全问题。等到攻击真实发生的时候，损失已经成为事实了。

代码的规范原则，是一个相对容易掌握的内容。高效的代码，也有很多成熟的经验可以学习。可是，代码的安全，却是一个攻易守难的问题。哪怕我们今天知道了所有的攻击和防护方法（这当然不可能），如果明天出现了一种新的攻击手段，而且全世界只有一个人知道，我们的系统都存在潜在的安全威胁。

编写安全的代码，需要掌握复杂的知识，而且需要大规模的合作。我们之前提到过三道槛，具体展开来是这样的：

我们要想掌握安全编码的技术，熟练修复软件漏洞的实践，需要先过三道关。

第一道关，是意识（Conscious）。也就是说，要意识到安全问题的重要性，以及意识到有哪些潜在的安全威胁。

第二道关，是知晓（Awareness）。要知道软件有没有安全问题，安全问题有多严重。

第三道关，是看到（Visible）。要了解是什么样的问题导致了安全漏洞，该怎么修复安全漏洞。

在意识、知晓、看到这三道关面前，我们要打开自己的视野，保持强烈的好奇心，从全世界范围内学习成熟的经验、先进的技术以及最新的进展。

其中，最重要的资源是 NIST 提供的[安全漏洞数据库](#)。这个数据库的使用方式有两种：第一种是了解自己的系统有没有最新的安全漏洞；第二种是学习最新的安全威胁的攻击方法和防范技术。

## 一起来动手

我们今天的练手题，就学着使用 NIST 的[安全漏洞数据库](#)。请你从这个数据库里，选择一个或者几个安全漏洞，试着看一下你的系统有没有类似的安全威胁？这个安全漏洞的攻击方式是什么样

的？这个安全漏洞的问题出现在哪里？该怎么防范？

欢迎你在留言区留言、讨论，分享你的阅读体验。

如果你觉得这篇文章有所帮助，欢迎点击“请朋友读”，把它分享给你的朋友或者同事。



©

 一手资源 同步更新 加微信 [ixuexi66](#)

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。

Ctrl + Enter 发表

0/2000字

提交留言

精选留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。

...