

42 | 纵深，代码安全的深度防御

2019-04-10 范学雷

代码精进之路

[进入课程 >](#)



讲述：刘飞

时长 11:21 大小 20.79M




前面我们聊了保持代码长治久安的两个策略，代码规范和风险预案。这一次，我们接着聊代码安全管理的另外一个策略：纵深防御。

说起纵深防御（Defence-in-Depth），我们最常想到的是军事战略。在军事上，这个概念指的是通过层层设防，以全面深入的防御来延迟敌人的进攻，通过以空间换时间的方式来挫败敌方的攻击。这有别于一战定胜负的决斗思维。决斗思维需要集中所有的优势资源在最前线，一旦前线失守，整个战争基本就宣告结束了。

信息安全的攻防，有一个很重要的特点，就是不存在没有漏洞的防线。按照决斗思维部署的信息安全防御体系，也许仅仅只能是个心理安慰。事实上，现代网络安全防御体系和应用架构，不管你是否意识到，已经在广泛使用纵深防御的思想了，或多或少，或明或暗。

评审案例

我们一起来看一段 OpenJDK 的代码修改。其中 wrap() 方法的传入参数 key 是一个不能泄露的密钥，而 key.getEncoded() 导出这个密钥的编码，以便进行下一步的加密操作。有时候，密钥的编码可以等同于密钥，也是不能泄露的。你知道这样修改的必要性吗？

 复制代码

```
1  byte[] wrap(Key key)
2      throws IllegalBlockSizeException, InvalidKeyException {
3      byte[] result = null;
4 +   byte[] encodedKey = null;
5      try {
6 -       byte[] encodedKey = key.getEncoded();
7 +       encodedKey = key.getEncoded();
8       if ((encodedKey == null) || (encodedKey.length == 0)) {
9           throw new InvalidKeyException(
10              "Cannot get an encoding of " +
11              "the key to be wrapped");
12      }
13
14      result = doFinal(encodedKey, 0, encodedKey.length);
15  } catch (BadPaddingException e) {
16      // Should never happen
17 +  } finally {
18 +      if (encodedKey != null) {
19 +          Arrays.fill(encodedKey, (byte)0x00);
20 +      }
21  }
22  return result;
23  }
24
```

这个代码变更，是对临时私密缓冲区的更积极的管理。

案例分析

我们知道，如果一段存储空间不再使用，一般而言，操作系统或者应用程序仅仅是“忘记”或者“删除”这段存储空间的索引，并不清理存储空间里的具体内容。我们常说“释放”一段内存空间。我觉得“释放”这个词使用很贴切。释放后，那段内存空间还在，模样也没有变化，内容也没有什么变化，只是被释放了，被丢弃了。

上面的例子中，`encodedKey` 是一个临时变量，定义它的方法调用返回，`encodedKey` 就被释放了。在这段代码变更之前，`encodedKey` 这段存储空间里的数据，在释放前和释放后，并没有变化。至于这段空间里的数据什么时候被覆盖，则完全依赖于 Java 和操作系统的内存管理机制，以及后续的内存使用。这种不确定性就存在一些隐患。

一段内存空间被释放后，由于这段内存空间的内容还在，这些内容就有可能被未授权的用户拣到、看到。如果这些内容是私密或者敏感的信息，比如密钥、口令、社会保障号码等，那么它们的泄露就是严重的安全事故。

假设还有一个程序，该程序分配了一段内存。那么，这段内存里可能就有别的程序释放、丢弃的内容。这个程序就有可能分析、转存、打印这些内容，进而造成上一个程序的私密信息的泄露。

我们在前面讲过整数的溢出，如果可以远程地制造整数溢出或者其他类型的内存溢出，这段内存空间的信息也可能会被泄露。

实际应用中有许多提高内存使用效率的技术，比如说缓存、虚拟内存、闪存、内存管理技术等等。这些技术在提高效率的同时，也增加了系统的复杂性，加剧了诸如内存溢出这类风险的破坏性。

高效率，是一个让人不懈追求的目标。为了高效率，对于大部分数据的释放，我们可以采取撒手不管的策略；为了安全，对于小部分敏感数据的释放，我们需要采取非常保守的策略。

敏感数据归零，就是其中的一个保守策略。

敏感数据归零是不是可以绝对避免敏感数据被泄露呢？当然不是，敏感数据归零也有很多解决不了的问题。比如说，对于上面例子中的 `encodedKey`，理论上还有以下的风险：

1. 在 `encodedKey` 归零之前，发生了内存溢出，`encodedKey` 有可能包含在被泄露的内存信息中。
2. 在 `encodedKey` 归零之前，底层的内存管理技术拷贝了 `encodedKey` 所在的内存块，当然也拷贝了 `encodedKey` 的内容。这时候，`encodedKey` 归零，有可能并没有清除拷贝的内容。
3. 在 `encodedKey` 归零时，由于不可控的编译器优化，`encodedKey` 的归零操作并没有真正地及时执行。

看起来真的像筛子一样，到处都是窟窿。那么，敏感数据归零到底还有什么意义呢？敏感数据归零虽然存在这样那样的问题，但是已经显著地降低了我们上面所说的风险。如果没有及时把敏感数据归零，风险会更大。**敏感数据归零是纵深防御体系中，非常具有深度的一个防线，但并不是唯一的防线。**

怎么设计和部署纵深防御体系呢？这是一个无比巨大的话题，我们没有办法几篇文章就交代清楚。下面，我们就尝试来理清楚这背后的逻辑。然后，你可以按照这些逻辑，去寻找相关的技术和实践，把它们运用到你的项目中去。

防线，攻击路径的纵深

一个有效的攻击，必须处于一个特定的攻击场景中。对应的防御，就是阻断攻击者到达或者创建这个特定的场景。比如说，上面的例子中，敏感数据的泄露需要攻击者能够访问敏感数据所在的内存。**为了设置纵深防御体系，我们需要在离内存十万八千里的地方就开始布防。**

为了形象一点，我们来看看一个有着“八道防线”的防御场景：第一道防线就是没有人可以接近存放这台计算机的街区，除了居住在街区里的人；第二道防线就是没有人可以接近存放这台计算机的建筑物，除了工作在建筑物里的人；第三道防线就是没有人可以接近存放这台计算机的房间，除了可以在该房间工作的人；第四道防线是没有人可以登录这台计算机，除了该计算机的操作者；第五道防线是没有人可以安装卸载计算机上的程序，除了该计算机的系统管理员；第六道防线是没有人可以访问内存空间，即便是系统管理员；第七道防线是没有人可以查看敏感数据，不管是谁；第八道防线是如果有人查看了敏感数据，或者泄露了敏感数据，隐私保护法会在前面等候。

这么严防死守，敏感数据还有可能被泄露吗？有可能。遗憾的是，无论是在理论上还是在实践上，还是会有人冲破这些防线。因为每一道防线都不是完美的，每一道防线都有天然的漏洞，而且每一道防线都需要执行才能发挥作用。这确实让人不满、不安。甚至有人说，这种基于攻击路径的布防已经老掉牙了，因为人们发现，世界上 80% 以上的攻击都是由于应用程序的漏洞引起的。可是，要是没有这些防线，实际有效攻击的数量一定会有数量级的爆发性增长。

机制，发挥防线的作

任何一道防线都不会自动发挥作用，除非我们设置了让防线良性运转的机制。比如说，针对第一道防线，我们有什么办法让只有居住在该街区的人进入该街区？谁来守卫这道防线？谁来检查、监管这道防线的有效性？如果非街区的人进入，应该采取什么措施？

还记得我们在[第二篇](#)文章里，谈到的优秀代码出道前需要经历的重重关卡吗？这些关卡是软件质量保障的一部分，也是让防线机制良性运转的一部分。比如我们上面提到的第七道防线，没有高质量的代码，这道防线的质量就是值得担忧的。

你不妨想想看，对于每一道防线，都应该设置什么样的机制？每一道防线都考虑**决策、执行、监督这三项权力的分配**，以及**计划、执行、检查、纠正这四项操作的实际执行**。这可能需要花费很长时间，也绝对不是一件容易的事。我相信，把这些问题弄清楚，理明白，哪怕只是针对其中的一道防线，都是一个了不起的成就。

多样，加固防线的双保险

最后，再给你介绍一个防护利器，那就是增加防护的多样性。什么是防护的多样性呢？

想一想你的车有没有安装警报器？如果说车门锁和点火锁是两道防线的话，那么安装警报器就可以增加防线的多样性。警报器是独立于锁的另外一种形式的防护技术。安装有警报器的车不仅试图阻止陌生人使用这辆车（锁），还警告试图使用这辆车的陌生人（警报）。这就是防护的多样性。

比如上面案例中的 `encodedKey`，如果代码再长一点，我们就需要在最早的时间执行归零操作，而不是等待系统的垃圾回收机制发挥作用。这也是多样性的一点点体现，它需要时间和空间上的双重考虑。

这里我要特别提醒你，一定要注意**多样性之间的独立性**。我们再来看小区门禁这个例子。一个封闭小区的入口不仅有门禁系统，还有门卫人员。这也算是多样性的防护措施吧。可是，我见到的封闭小区的管理，几乎门卫人员总是可以开启门禁系统。门禁系统和门卫人员并不是完全独立的。有些时候，这不是强化了防线，而是弱化了防线。只要和门卫人员搞好关系，门禁系统就是虚设。在计算机系统中，系统管理员、数据库管理员的权限就有点像门卫这个角色。**如果信息系统防护的多样性之间不能独立，多样性的防护实际上可能会产生多样性的漏洞。**

小结

通过对这个评审案例的讨论，我想和你分享下面两点个人看法。

1. 没有防御纵深的信息系统，其安全性是堪忧的；
2. 一个防御体系，需要考虑纵深和多样性，更需要确保防御体系良性运转。

一起来动手

把纵深防御理念用得最讲究的，在我的见识范围内，我认为是核防护的设计，毕竟这是关乎巨大人群生死存亡的大事。如果要梳理基本概念，树立谨慎保守的防护理念，我建议阅读下核防护的一些规范。你可以从[核电站通用设计准则](#)开始。

那么，怎么把理念落实到应用软件呢？我建议你阅读 Oracle 的[纵深防御指南](#)。这份指南虽然是为 Oracle 数据库准备的文档，但是其中涉及的很多思想、方法和技术同样适用于其他的软件和编码领域。非常难得的是，这个纵深防御指南罗列了不同维度的很多检查点。我们可以使用这些检查点来查验我们的软件设计和实现。

聊起纵深防御这个话题，就真的不是几篇文章可以说的完的。希望今天的内容能够抛砖引玉，欢迎你把自己的经验和看法写在留言区，我们一起来学习、思考、精进！

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。



代码精进之路

你写的每一行代码都是你的名片

范学雷

Oracle 首席软件工程师
Java SE 安全组成员
OpenJDK 评审成员



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 41 | 预案，代码的主动风险管理

下一篇 43 | 编写安全代码的最佳实践清单

精选留言 (3)

写留言



天佑

2019-04-10

1

核电站通用设计准则,访问不了, 显示不允许访问。。

展开

作者回复: 😊



hua168

2019-04-10

1

我们怎么防止还是会有想不到位的地方, 那么我可不可以做一点监控或者一些策略。

- 1.做监控是为了方便跟踪, 知道对方是怎么进来的
- 2.做策略是补防守不周全的情况, 比如前面再加多一层安全过滤层, 做一些行为判断
比如定义一些异常访问, 如包括一些特殊字符, 异常行为

...

展开

作者回复: 有些公司的商业模式安全很重要, 早做的好。

工作量这个标尺, 太耽误事! 没有工作的工作量最小。做最简单的事情, 把简单的事情做好。别做大量的事情, 把事情做的闹心。不需要现在就做的事情不去做, 我们就有时间把少量的事情做好了。



松花皮蛋me

2019-04-10

1

老师我觉得专栏的安全方面内容太多, 但是技巧类的太少了

展开

作者回复: 😊, 技巧类的太多了, 而且我懂得极少, 所以写的也少。

