重新探访JAVA安全体系结构

Java平台JDK 1.0于1995年发布，带着一个简单的可有可无的“沙盒”的安全模式。Li Gong在1996年加入了SUN公司的JavaSoft部门，带领重新设计安全体系结构，并于1998年发布了JDK1.2， 现在部署在从桌面应用到企业以及java移动端的众多系统和设备上。  
 本文回顾了几个从设计和工程的角度，以及为这些科学家很少被培训一些棘手的业务挑战最困难的技术问题。LI回顾了他能够深入到旧的笔记和重现自己的记忆的四个场景，分别是：2002年在伯克利的信息安全经济学研讨会；2003年在华盛顿UW/ MSR / CMU暑期学院的史蒂文森；2009年在檀香山谈香山的ACM计算机安全应用大会上; 和去年五月在英国剑桥大学计算机实验室的研讨会。

虽然安全架构师不参与业务， 但重要的是他们清楚谁是他们的客户。这些客户很少直接建立个人终端用户，不直接使用操作系统， 但他们往往是最终的受益者。

安全架构师的大部分工作是针对开发应用的程序员，java也不例外。设计目标是帮助程序员知道其代码的目的是什么，更具体地说，使最容易且正确的编写成为最常见的情况，并减少编码错误或错误的风险。这样，在Java安全架构1的四个属性一般应适用：  
**可用性。**

被广泛使用在市场上被接受，该架构必须易于使用，并适合于书写的各种应用程序。

**简易性。**

为了激励在架构的正确性充满信心，一定是容易理解的关键设计性能和分析落实。

**充足性。**

该架构必须包含所有必要特征和模块为支持更高级别的安全性的要求。

**适应性。**

设计必须轻松演进，适应新的需求和市场变化。特别是，应避免过度设计，限制可扩展性。

在事后看来，这些指导原则是至关重要的。在原来的JDK 1，安全机制都是关于特殊外壳代码的沙箱内与外。看似简单的架构，矛盾的是复杂的设计，脆弱的代码，和众多的安全漏洞。在JDK 1.2中，安全性是设计是通用的，系统的，简单的思维，这导致在一个更强大和实用的平台。我们击退竞争设计，是专门为从Netscape浏览器的使用。我们的设计不仅广泛的范围，包括台式机，服务器，嵌入式和移动设备，而且还具体到足以使程序员建立以浏览器为中心的应用程序。稍后我会回到这些主题。

**猜猜谁来吃晚餐**

设计的愿望是确保java代码执行预期的无不良影响。这个目标有三个组成部分。首先是要确保只有有效的java代码被接受；这是本节的主题。其次是确保预期行为发生的设计，这通常是通过测试，这理解，因此这里不做过的解释。第三个是防止不好的行为，如本不应该被允许的关键数据访问 ，这是本节后面将会讲到的最小特权原则。

然而，另一个经常隐含这样的要求是，所有的制衡必须做的相当快，这意味着该系统具有的性能特性要与，没有安全机制在所有的系统相媲美。这里的威胁模型主要侧重于不可信的代码，可能从事恶意行为。该保护机制的目的是阻止那些恶意的行为; 它也有助于减少良性编码错误的风险，虽然不能指望抵御所有错误的编程习惯，如不确认，可能导致SQL注入攻击查询。

通常，应用程序由JAVA代码编写，被编译成独立于平台的Java字节码，然后由JVM （ Java虚拟机）执行。它可以直接编译Java源代码转换成特定机器的本地代码（如x86系统） 。因为编译的本机代码绕过了Java机制，并且不能与完全Java平台内除了通过允许或禁止机代码访问和执行的方式来处理这种情况下不再进一步讨论在这里。也可以直接写Java字节码，尽管大多数人选择不练这个特殊的艺术。在任何情况下，即使是由编译器生成的字节码也可能是不被信任的。事实上，肯·汤普逊与他著名的1984年图灵奖演讲也已渐行渐远， “反思信任信托”，说，“你不能信任任何代码，除非它是完全由你自己编写的。 ”因此，在JVM必须能够决定是否一块字节码是有效的，可以接受的。  
 Java字节码（操作码）的每个单位是正好一个字节长而被很好地定义。一个真实的编译器需要有效的Java源代码，并且产生字节码的一个序列准确refects的源代码的意图，并在同一时间保持Java语言的固有性质，如类型安全。而另一方面，字节代码的恶意生成的序列，可能不符合任何有效的Java源代码并且可以故意打碎的语言特性，从而进行安全攻击。  
 一个呈现一系列操作码是否是“有效”基本上与输入验证问题的一种形式。假设在JVM指定输入任何在1至9范围内的整数均为有效输入 ; 这种输入验证的作用是微不足道的。在现实中，包含任意序列的操作码的输入空间是无限大的，有效操作码非常稀疏地分布在这个输入空间中。因为在目标代码运行时没有简单的验证试验，Java运行时系统在多个阶段，以不同的技术，在不同的地方执行字节码的验证。字节码校验静态地检查传入的代码。一旦代码是在系统内部，部署于整个JVM的类型安全机制将发现和制止非法代码。所有这些操作是复杂的，并且，在不存在以整个系统的形式验证时，就无法知道是否能确保所有无效的代码被发现。

因此，对于任何的运行时系统，该系统与高级语言编译的可执行代码的交易之一，难题之一是要确保从外部源收到的代码是有效的输入。对于大多数编程语言，这简直是不可能的。  Java的跨平台的字节码，使这项工作变得有可能，但它仍然是非常困难的。布赖恩伯沙德和他（当时）的学生Emin Gun Sirer 在华盛顿大学提出了字节码破坏者的概念。他们建立了一个自动化系统来生成随机序列的操作码并在任何的Java运行时系统抛出; 他们将会看到系统是否中断，然后对结果进行分析，以找出在Java实现中的缺陷。这项随机和自动化的方法是一个令人惊讶的低成本而有效的工具，它被JavaSoft的团队迅速采用。

**谁动了我的奶酪？**

现在问题来了，如何防止意外的不良行为。例如，当一个Java应用程序或小应用程序触发一个本地文件的访问请求时，请求是否应该被授权呢？嗯，这取决于。如果请求是读取包含个人信用卡信息的本地文件，则该请求可以被授权或者被拒绝，依赖于是否存在安全策略或用户偏好授予。如果请求是读取为12点宋体类型的字体文件，使得一个文本文件可以根据字处理器规格被显示，则它几乎总是应该被授予，因为，字体文件的结构是无害的。请注意，应用程序从不直接打开文件。他们通过调用Java平台的文件接口来进行这些操作。这些接口没有内置安全性的概念，但他们（或他们的设计师）知道，文件操作是敏感的，所以他们最好打个电话与SecurityManager做个磋商。

现在问题又来了，SecurityManager来到了现场但也没什么可说的。例如，在该系统中编写可以访问该字体文件的显示代码是完全没问题的，但通常它是一个坏主意，用于获得直接访问系统字体文件，因为这些文件可以被任意改变，这可能导致将来的显示问题。SecurityManager难以区分这两个场景，更不用说不明确的变化了。

在jdk1.0/1.1中，孤注一掷的沙箱安全模型或多或少的做如下工作。SecurityManager查询方法的调用历史，如果所有的代码都是本地的（即，没有因远程加载而不受信任的应用程序），则该访问请求被许可。如果一个应用程序在调用链中，则该请求会被拒绝，除非是系统代码请求访问该文件。除非该代码不应该访问字体文件，小程序代码回调情况除外；除了系统的调用返回（小程序）不应该真的访问字体文件，等等。此外，线程、异常和其他结构的构造，或断开执行上下文？你脑子里会生成一个图像的。

从根本上说，试图猜测一个程序的意图是不切实际的。要求程序员显式地声明他们的意图，这样会好很多， 稍后我们将看到。

更糟糕的是，对SecurityManager的实现实际上不是通过此逻辑推演运行，它也不能。相反，基于一些经验法则和java系统的一个特定实例，简单的要数距离的方法调用的次数和最近的小程序代码之间，并试探性地提出一个请求是否应该被授予一个决定。现在应该是显而易见的，这一设置意味着，当系统的一部分被改变，启发式可能成为错误的，因此必须进行调整，无论是正确的启发式或完整的第一位，无论使用的方法，以扩展系统，包括一个新的概念，如用户/校长的访问请求的决定。这个脆弱的设置是许多安全漏洞的来源。

JDK1.2的安全是被完全重新设计过的，采用了著名的但却几乎没有被实践过的最小特权原则。对所有的代码都一视同仁，对每个代码段都给出一组显式或隐式的权限（访问权限）。执行中的每一个点，如果每段代码在调用链上有足够的权限来请求访问，则授予访问权限。换言之，有效的特权集是所有特权集的交集，这是一种最不优惠的原则。此外，与安全相关的上下文信息可以被封装和传递，因此无法通过产生新的线程会抛出异常来骗过系统。所有这些假设，当然，实现安全机制的代码本身是安全而且正确的。

有一段代码，它是这样的，显示库可能需要一次又一次地访问字体文件-------可以在告诉安全系统忽略代码之前 显式地声明单方面行使自己的特权，。这个设计让程序员，特别是那些编写系统和库代码的程序员，在执行敏感操作时要显式地声明他们的意图。这类似于Unix中除了setuid的功能，与其为整个程序启用系统级别的高特权， 不如是在Java的特权模式只持续只要特权方法调用的持续时间。注意，如果此特权代码调用权限较低的代码，有效权限集仍然会因为最小特权原则限制。

显示声明可能在一开始就会变得繁琐，但程序员可以放心，他们的代码不会无意中被削弱安全性。此外，大多数的项目不需要调用他们的特权，所以我们在大多数情况下都能在安宁的情绪中简单地编码。请注意，为了让程序正确地工作在所有场景，准确地计算出哪些权限需要声明，这对于程序员来说并非一件容易的事。JDK 1.2的设计其实不支持细粒度的权限声明。一个明可以激活所有与该代码关联的特权。

这里的主要教训是成为系统性的东西比特别的设计更简单且更具鲁棒性，但不是每个人都明白这一点。到JDK1.2开发结束，在对整个代码库的安全审计期间。我们发现，一个在SUN 公司为JDK工作的工程师故意复制然后修改系统代码库的代码，这样，他自己的库代码将不必理会一个明确的权限声明。此举可能会使他的工作变得容易些但会导致严重的安全漏洞，但如果他的罪行被发现，这对java平台的所有用户来说都是一个严重的安全破坏。

在这个领域内还留存着许多困难的问题。其中的一个问题是，是否可以有效地完成最小特权的计算，特别是具有非常复杂的安全参数，例如，复杂的访问控制策略，许多不同类型的访问权限，和一个复杂的执行环境。另一个问题是，将不同的权限分配给系统的其他部分，创建了不同的代码。例如，优化了JIT（及时）编译器必须符合额外的安全要求。

一段代码可以单方面地显式地声明行使自己的特权，告诉安全系统忽略在这之前的代。这个设计让程序员在执行敏感操作时，可以显式地声明他们的意图。

然而，另一个长期的问题是安全策略管理和部署的实际方面。一个更理论化的问题，但也很值得深思的是，能够（或不能）使用这样的安全策略——用最少的特权模式，使用在JDK1.2中定义的传统的访问控制权限类别执行。

康奈尔大学的弗雷德·施耐德开发了一个有趣的概念叫做内联引用监视器和证明，它可以表达和执行任何政策的执行实施监控系统。

尽管有这些困难的问题，一个让人欣慰的想法可能是—— 12年或者更久后，最小特权原则在JDK 1.2架构已经经受住了时间的考验，可能挽救了不计其数的变成安全失误的编码错误。

**认真的重要性**

许多其他的技术课程是值得周期性地复习。例如，你应该非常明智地使用空，因为你不能改变任何事的行为。在**JDK 1 / 1.1**的某些情况下，类加载器或**SecurityManager**可以为空，这使得它很难更细粒度的改造设计。

另一个例子是，在运行过程中，**java**变成静态代码到新的对象。这个过程实际上包含了一个单独的步骤：定位代码描述，将其定义为活对象。第一步应该是开放的，可扩展的性质，因为这两个运行时系统和应用程序应该能够指定所需的位置获得代码。第二步，另一方面，必须严格控制，使唯一可信任的系统代码可以处理创建对象的工作。不幸的是，这两个步骤都超载成一个单一的方法，在全有或全无的模式运作良好，但引起了很大的困难时， JDK 1.2变成一个更细致入微的世界。

另一个问题可能会让很多人吃惊**:**严格来说**，Java**不能保证连续的顺序执行指令。一个简单的原因是异常可以被抛出**，**导致执行线程绕道**(**和可能永远不会返回**)**。一个补救方法是使用语句如**Try / Finally**强迫回归。在更极端的情况下**，**例如当实际的物理机器的内存耗尽——**Java**运行时系统的行为是未定义的**，**当然远不及被破损安全。这些情况进一步复杂化的事实，是因为许多关键**JVM**功能**，**包括一些安全**，**是用**Java**编写的，所以系统某一部分的问题很容易影响系统的另一部分的正确性。对于所有这些设计挑战和选择**，**请参阅**Java**安全**book2**和最新的**Java**文档。

本文的其余部分地址完全意想不到地挑战了以前工作经验仅限于学术界的世界的人。但在真实世界的项目中，科学家和工程师们被训练来解决技术问题**，**，尤其是像社会和政治这样具有全行业影响的，java也是如此。在**JDK**大约**30**个月的工作**，**我参加了**1000**个右的会议，花了**300**多页的笔记。人们在事后很容易忘记战区的气氛**，** 特别是周五消防演习。**(**外部的**)**安全研究人员常常会在周五告知我们新发现的安全漏洞，但直到周一中午才给我们到找出一个补丁和响应**，** 中午的时候他们会告诉《纽约时报》、《华尔街日报》和其他媒体。有时在我们推出了补丁，**java**授权（包括**IBM**、微软、网景、和许多其他的）时，记者就泄露了消息，我们只能猜测他们有在补丁到位之前宣传安全漏洞的动机。

还有各种其他的耗费同样时间和精力的干扰**，** 例如美国在基本加密（因为为已经不安全）上的出口管制条例， **RSA**公钥技术专利**(**因为过期**)，** 代码模糊， 已经**Java**在电子商务和**Java**智能卡上的应用**， JavaOS**等。

为了确保我们走在正确的道路上，我们邀请了少数学术界和工业界的专家

（包括麻省理工学院的杰罗姆Saltzer和DEC系统研究中心迈克尔·施罗德，最小特权纸的原主要作者），并召开了正式的Java安全咨询委员会，提供定期的和有价值的反馈，重新构建进展。我们也从很多渠道收到了宝贵的建议，主要是学术研究人员和行业合作伙伴，但并非所有都是友好或被我们接受的。少数几个研究巨头希望他们的替代设计被纳入到**java**平台，而且还各种威胁我们。

网景是一个独特的故事。这是最流行的浏览器，而且包含了Java，因此是一个重要的合作伙伴; 它对于应该在哪些方面引领JAVA有自己的想法，而那些野心让双方的关系变得困难。技术层面上，在安全领域的主要争议是Netscape公司认为Java基本上只是一个浏览器组件所以安全机制应该是面向用户的浏览器，而我们的愿景是Java是一个通用的编程平台，应该照顾到所有种类的用途，包括浏览器和服务器端应用程序。

在工程级别上，网景公司每三个月就创新并投产一个新版本，而Sun/JavaSoft需要一到两年时间通过官方的JDK平台更新版本。JDK和Java代码之间的分歧在网景公司变得过大，网景和JavaSoft的总裁邀请IBM来执行保密和有约束力的仲裁。经过几个月密集的实况调查、代码收集和用户反馈评分，IBM于1997年10月15日在离JavaSoft只有一个街区的Java大厦召开了仲裁会议，宣布JavaSoft取得了胜利。

这么多年后再回顾， 我可以看到至少有三个对Java安全有持久影响的工作。最明显的是，新的安全体系结构帮助Java程序员将他们的应用程序构建得更安全并且当代码发生错误时减少风险提供了更好的支持。第二，我们提高了其他人对任何新的语言或平台必须考虑类型安全、系统安全、和最小特权原则的认识，因为我们已经证明，即使在大规模的商业环境中这些都是可以实现的。最后，在Java中的安全结构增加了开发人员的安全意识，为数以千计的开发人员将这种知识带到了其他编程语言和开发平台。

**致谢**

这里我要感谢卡内基-梅隆大学的Jeannette Wing，斯坦福国际研究所的Jeremy Epstein 和Peter Neumann，以及谢剑桥大学的Ross Anderson 和Robert Watson邀请我参加这次关于JAVA安全的讨论。

我很感激 和 在ACM鼓励我将剑桥的谈话发表，通过匿名的方式得到了周到的评审。当然，我对所有关心、帮助、支持JAVA安全项目的人都负债累累。