**大 连 理 工 大 学 本 科 外 文 翻 译**

**零日之前：**

**安全软件工程的投资模型**

**The Days Before Zero Day: Investment Models for Secure Software Engineering**

学 部（院）： 软件学院

专 业： 软件工程

学 生 姓 名： 侯坤鹏

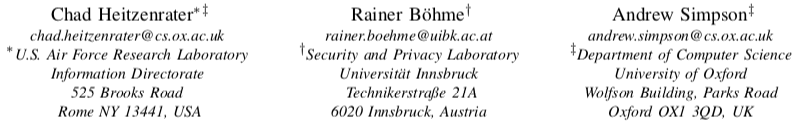
学 号： 201592146

指 导 教 师： 马瑞新

完 成 日 期： 2019.2.20

大连理工大学

**零日之前：安全软件工程的投资模型**



**摘要** - 虽然大多数安全实践和支出都侧重于后续的产品开发和企业方法，但有些人试图将安全性的重点从我们管理的网络转变为我们构建的系统。新兴的安全软件工程（SSE）团队力求识别和支持基于传统软件工程解决漏洞的引入问题的方案，以此在安全问题出现之前阻止它的日益增长。人们普遍认为，这种方法不仅有望减少安全问题的出现并减少安全事故，而且它们也是一种有效的安全投资，可以降低整体安全支出。虽然目前正在采取许多措施来实现SSE，但尚未对经济方面的考虑进行处理。我们提出了一个初始模型，该模型将SSE投资作为降低防御者对漏洞的不确定性的一种手段，同时提高攻击者的成本。这种方法被实现为传统安全模型的必要流程，我们使用（部署后）安全投资的迭代最弱链接（IWL）模型来演示如何在系统的生命周期内优化防御性的安全投资。结果表明，安全投资回报率增加 - 安全软件过程回报率（ROSSP） - 以及部署后成本降低。我们希望这种模式为统一前后的安全投资更全面处理铺平道路，，从而更全面地了解软件系统的安全性。

**1.简介**

计算机安全已成为安全产品的代名词。 作为消费者，我们安装了防病毒软件，修补和更新我们的应用程序，并配置我们的防火墙和路由器。企业投资于一系列的供应商产品和服务，通常会造成解决方案的混杂。 安全行业本身可以被描述为一系列“下一步”解决方案，安全专业人员接受一系列安全设备的培训。安全投资已经日渐保守- 优化资源支出以解决在很大程度上可以预防的问题，所以一些人将当前环境定性为安全思想的“黑暗时代”。

这并不是说系统级预防性方法不是安全性的必要组成部分。在实践中，不同的系统必须集合在一起形成体系，紧急安全情况下是没有多少方案可以预测的。但是，仅仅注中企业安全投资是一种不完整的方法，无法反映漏洞的来源和发现方法的固有缺陷。对于那些希望绕过外围防御的人来说，应用程序越来越成为“进入系统的一只利剑”，这使人们越来越相信，解决计算机安全故障的正确环境正在发展中。然而，信息安全投资的模型还没有跟上这种观点，继续专注于保守的后期开发和优化。

在本文中，我们介绍了一个包含软件开发过程的安全投资模型。安全软件工程（SSE）包含了被广泛视为漏洞引入的最佳补救措施，但这些措施很少受到信息安全投资界的关注。我们的模型为从业者提供了理解系统生命周期中SSE投资的机制，以最大限度地提高安全性。

在第2节中概述了相关的软件工程和安全投资概念后，我们在第3节中定义了SSE投资建模所解决的问题。接下来，我们概述了迭代最弱链接（IWL）安全投资模型，并以此作为我们形成初始模型实例化的基础。我们的安全软件工程投资模型IWL-SSE将在第4节中介绍，并讨论模型的运行。第5节介绍了对更广泛影响的分析，其中我们介绍了安全软件过程回报（ROSSP）作为比较SSE决策机制的指标。我们在第6节中对未来的发展方向进行总结。

**2.背景**

在众多公认的安全原则中，项目规划中经常提到的一点是“安全是一个过程，而不是一个产品”。尽管如此，对系统开发所涉及的过程的处理通常被安全投资界视为理所当然或被忽视。 我们的模型建立在这样的前提下，即在系统开发期间对安全实践的投资对于真正安全系统的生产至关重要。 本节提供了进一步论证来支持这一观点的必要背景。

**2.1.软件工程过程**

系统和软件开发的基础是系统开发生命周期（SDLC）的概念。虽然这个术语通常被广泛使用 - “S”有时意味着“软件”，而不是“系统” - 美国国家标准与技术研究院（NIST）试图在[8]中编纂概念，指定安全实践系统生命周期的五个阶段：启动，开发/获取，实施/评估，操作/维护和处置。许多常见的安全问题，例如标准认证，配置管理和修补以及周边安全，仅涉及实施/评估和操作/维护阶段。

在SDLC的背景下，有个组织定义了他们的软件开发过程，可能跨越每个阶段，但通常集中在启动和开发/获取阶段。这些过程模型是软件开发的最基本方面之一，它管理开发活动的内容，频率，时间和范围（需求，架构，设计，编码，测试和发布）。软件过程的常见变体包括瀑布式，渐进式，渐进式/增量式，螺旋式，原型设计和敏捷模型;入门书籍，请参考[9]。考虑流程模型需要考虑组织和项目约束，包括考虑现有系统，项目范围（独立与产品系列的一部分）以及项目团队的构成。

除了为软件开发管理提供体系结构外，软件工程长期以来一直试图量化软件开发背后的经济学。 人们早就知道，在软件开发过程早期做出的投资决策更具成本效益[10]，实证研究证明了成本在系统寿命期间的影响（表1）。 这些数据通常是整个开发过程中投资决策的基础; 然而，由于几乎没有良好的安全指标，这种量化方法尚未提供对安全投资的相同见解。

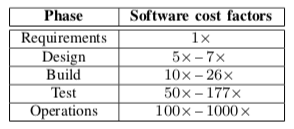


表1：各阶段补救的软件成本估算（来自[11]）。

**2.2.安全软件工程**

虽然软件工程的实践通常与高质量软件的开发有关，但是新兴的安全软件工程团体试图通过面向安全的练习来扩张特定的流程步骤。为此，有人说“防止在发布之前引入漏洞，而不是在之后修补漏洞是SSE应对的挑战”[2]。这种推理方式区分了“应用程序安全性 - 主要侧重于在编写软件之后确保软件安全”与“以网络为中心的安全方法更密切相关 - 以及软件安全性”，旨在“利用良好的软件工程实践并引入在软件生命周期的早期考虑安全性的思想”[12]。

这些实践的重点是识别和消除缺陷，设计错误和错误，这些都是项目实现中的错误[13]。盛行的安全软件工程常识认为，项目的弱点50%来源一漏洞，50%来源于程序错误 [14]。迄今为止，针对这些目的的软件安全性的尝试有两个方向：

* + 开发过程，主要侧重于使用过程和工具来减少项目实施的错误. 实践包括OWASP CLASP1，微软SDL2和Adobe SPLC3。 其中最广泛适用（也是大多数引用）的可能是SSDL-Touchpoints[5]，它规定了七种包括操作要求的实践。
  + 实践建议，旨在解决当前开发过程中的常见错误。其中包括规范性实践，如IEEE CSD十大建筑缺陷[15]和OWASP前十[16]，以及广泛的元模型，如建模安全元模型（BSIMM）[17]和分类整理常用实践的OWASP OpenSAMM [18]。 BSIMM是对不同组织内现有软件安全计划的研究，旨在为更广泛的软件安全领域提供信息。它被称为“软件安全当前状态的反映”[17]，确定了如滥用案例和特定于安全性的需求定义（在需求阶段），风险分析（在需求，架构和设计方面的实践和测试阶段），基于风险的安全测试（在测试计划中），代码审查（在代码阶段），渗透测试（在测试和部署）和安全操作（在部署之后）。

这些类别并不完全明显; 例如，BSIMM将Touchpoints 作为四个推荐的实践类别之一（以及管理，情报和部署活动）。 此外，它们有共同的局限性; 虽然对要采取的各种步骤有规定性，但对于为这些做法投入的努力量却鲜有规定。

**2.3.信息安全投资**

在信息安全投资领域，模型往往侧重于企业层面。这些模型考察了产品投资之间的权衡，而不是考虑与系统开发和部署相关的过程。这种关注限制了软件生命周期内的安全投资权衡因素，忽略了其创建和部署后的存在。当安全决策的影响无法考虑软件生命周期时，风险被忽略，选项受到限制，并且由此产生的系统在其接口处具有扩大的攻击面和固有的复杂性[19]。

我们的模型最好与其他模型相结合，使用多目标优化和游戏理论关系等工具，在各种环境中建立最佳安全投资;这些模型的回顾可以在[20]中找到。这种投资模型的理论基础可追溯到[21]。示例游戏理论模型包括[22]，[23]和[24]中提供的模型。在[25]和[26]中讨论了将投资指标应用于安全投资的问题。最近试图检验这些权衡的游戏理论模型，如[27]中的那些，提出了我们在模型的开发级别方面与企业级别考虑所需的更高级别系统安全性思想之间呈现出自然的交叉点。有些人已经试图通过可以最好地描述为预部署、系统级流程的定义来将这些考虑推到过程的早期。诸如安全属性评估方法（SAEM）[28]和适当且有效的信息安全指南（AEGIS）[29]等概念在系统级别考虑系统安全性，但仍然是后期开发步骤。这些方法通常基于风险，依赖于系统工件（设计，组件等）的存在来集中基于资产的决策 - 主要是在没有成本考虑的情况下。

综合考虑范围、成本和时间的挑战证明，迄今为止，在安全投资模型方面的研究存在局限性，同时这表明在基于软件的系统的整个生命周期内对安全投资的综合处理普遍缺乏。我们认为，必须为项目管理者提供适当的工具，以便对他们对安全软件过程元素的投资做出合理的决策。本文的目的是为了促进开发结合软件工程、安全软件过程和信息安全经济学等元素的工具的发展，从而更好的为我们提供可用工具。

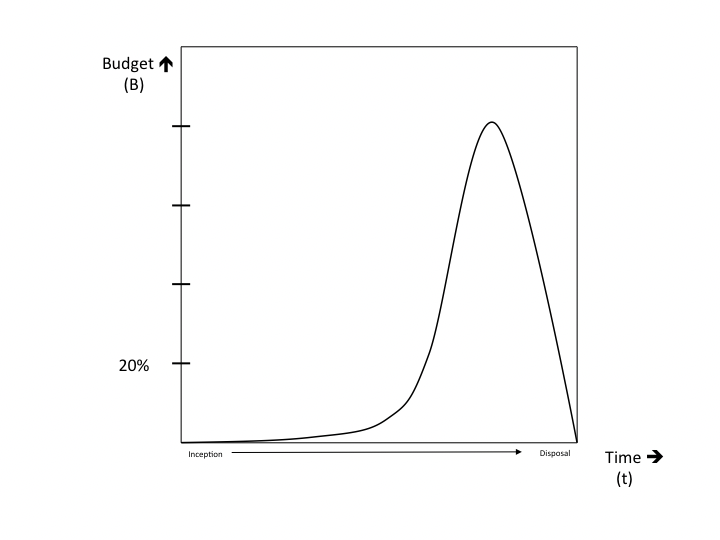
**3.问题陈述**

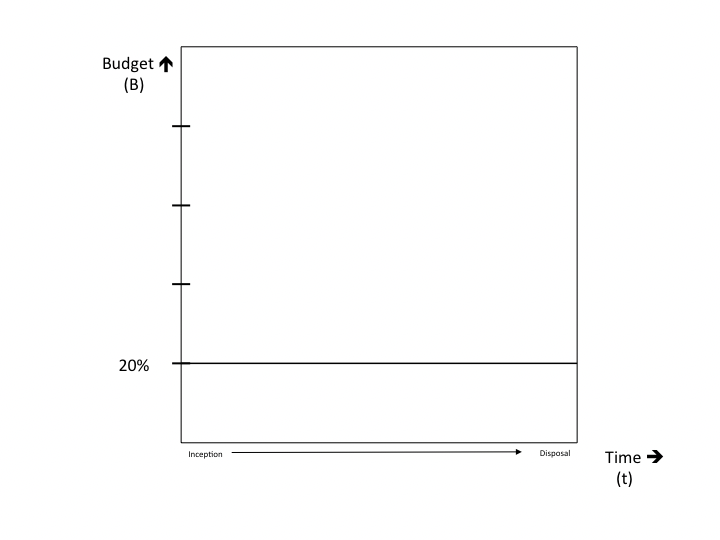
在当前的境况和SSE理想情况，一个合乎逻辑的问题出现了：是否有一个项目资源分配能够达到高效？

**3.1.软件过程投资**

软件系统的安全投资可以表现为分布在SDLC阶段的一系列决策，从Inception开始到系统的Disposal结束。然后，我们可以将系统设想为具有固定预算B，其中B的一些分配专用于在任何时间点t的安全性（表示为Bs）。将其投射到笛卡尔空间，我们发现Bs可以呈现多种形式;图1a-1d介绍安全投资的几种情况。

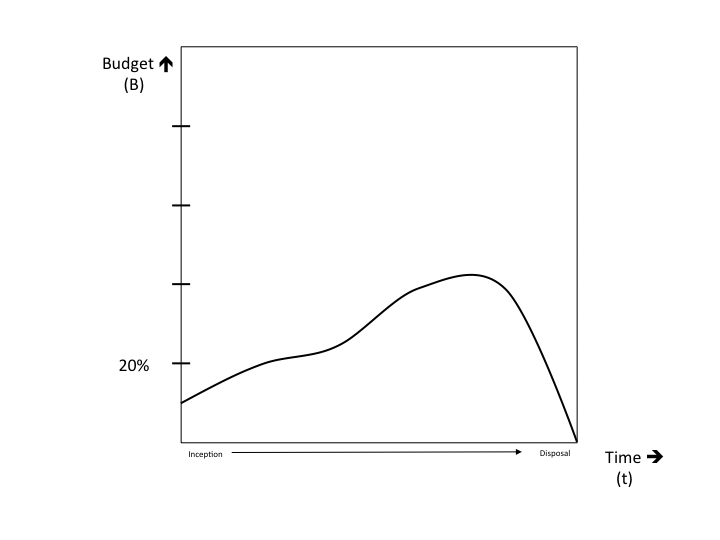
图1a也许显示了当前情况：在流程的早期阶段进行的安全性投资非常低，很快就会增加以消耗不断增加的项目成本份额。这种投资方法除了效率问题之外，还会将资源分配给安全性，以应对后期其他投资机会的潜在损害（例如新功能或代码维护）。图1b虽然受到更多控制，但可能不是有利的分配，因为任何特定阶段的安全投资具有相同的利益。许多人可能认为图1c是最有可能是有效投资的情景，随着安全问题 - 以及系统伪影 - 变得更加切实，安全投资也会相应的增加。然而，SSE可能会争辩说，如图1d所示的投资概况是最合适的。这反映了一种观点，即前期投资是实现安全性的最有效手段，减少了后期投资，例如：认证、修补和违规补救。这些描述中哪一个（如果有的话）反映了一个更有效的真是情况是本研究的重点。



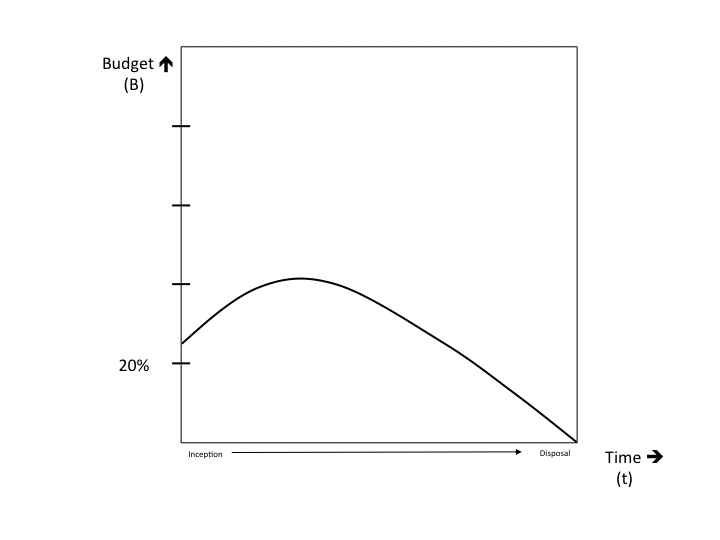


（a）名义描述，显示在生命周期后期（例如部署后）发生的大量安全投资。

（b）在项目过程中，将常数安全预算B作为每阶段预算B（此处为20％）的固定比例的名义描述。



（c）在生命周期早期开始并逐渐增加的安全投资的名义描述。



（d）在早期阶段对安全投资进行名义描述，导致后期阶段的投资减少。

图1：系统开发生命周期（SDLC）上的安全投资描述

**3.2. 假设和约束**

我们考虑问题倾向于将SSE原则应用于特定的开发环境、模拟软件工程过程、最薄弱环节模型。我们寻求在以下背景下解决对SSE的投资。

1）我们假设独立地进行新项目开发，所有安全投资决策都留给项目管理者，其影响仅限于个别系统。这消除了对接口，集成或现有框架改变投资决策的任何担忧。

2）我们假设一个分阶段的顺序开发生命周期，其中开发过程由程序管理员明确定义，控制和指导。这很容易被认为是瀑布式或增量型，具有定义的架构/设计和代码/测试阶段。虽然这里仅考虑两个阶段，但在第6节中对软件开发过程进行了的更广泛方面的讨论。除了简化模型之外，有证据表明这种模型在许多开发环境中仍然是主导方法[30] ，[31]。

3）我们假设开发人员参与开发的部署和操作，或者该信息可靠且真实地传达。信息的传递与安全决策所依据的过程伪影的不确定性直接相关，并且是我们模型的关键要素。这可能是最真实的软件工程环境之一，但在实践中仍然不常见。通过采用这样的背景，我们寻求在最广泛的意义上定义我们的模型，同时使其易于识别并且易于从业者采用和适应。

4）我们假设：一旦发现缺陷或错误，它就会被修复;制定的修正是正确的;并且每次都以相同的效率重复进行SSE活动的迭代。虽然有迹象表明情况往往并非如此[32]，但这种过程复杂性和次要影响的影响仍待研究。

我们专注于BSIMM的SDDL-Touchpoints领域内的实践：架构分析（AA），代码审查（CR）和安全测试（ST）。 这些活动中的每一项都是已建立的软件开发实践的一部分，SDDL-Touchpoint和BSIMM提供了一个增强常规实践的安全性重点。 然而，对于成功完成任何这些步骤或集体投资几乎没有指导意见。

**4.模型**

为了评估整个安全系统的开发，我们研究了如何在一般信息安全经济模型中表示SSE过程。这项工作的目标不是忽视或替换现有的结构和假设捕获各种安全方案的重要方面的模型，而是增加其构造。对于这个初步检查，我们选择使用迭代最弱链接（IWL）模型[6]作为建模构造，我们将通过添加SSE考虑因素进行扩展。 IWL提供了许多功能，使其成为这种补充结构的理想候选者，因为它侧重于基于软件工程核心的启动条件优化防御者决策：质量（以攻击者的成本形式）和关漏洞的不确定性。 SSE也显示出最弱的链接行为，正如IWL所反映的诱因那样：“软件公司中最粗心的程序员可能会引入一个关键漏洞”[6]。诸如“前10”之类的常见实践——驱动审查以及代码分析工具和规则集的使用随着“yesterday’s attack ”的实现而发展；它们旨在将重点放在最常见的错误上，通过越来越深入的审查和分析来推动安全性和成本。

现对IWL进行审查，以确定我们的SSE投资模式所需的条件。

**4.1.迭代最弱链接模型概述**

IWL的核心是攻击是“不可知的，因此无法提前”[33]。这导致了一个模型强调随着时间推移的动态适应性投资的定义。作者试图通过关注以下三个关键特征来捕捉安全投资的迭代性质。

•后者对哪些组成部分最弱的不确定性是投资决策中的一个关键考虑因素。

•防御作为攻击和防御连续最弱链接的迭代过程。

•对策可以表示为相互依赖（使得该模型中信息安全投资的边际收益递减）。

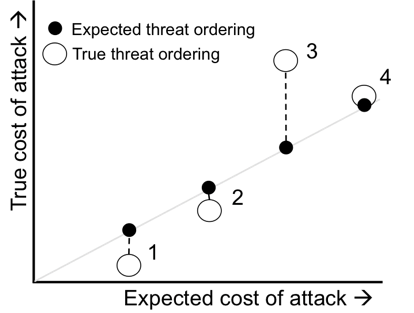
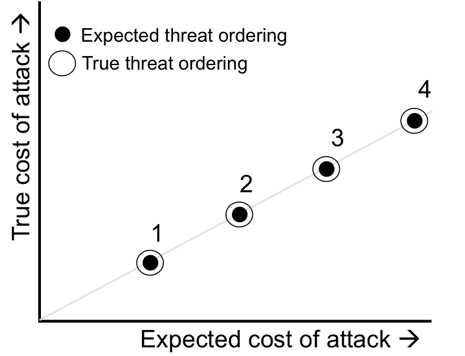
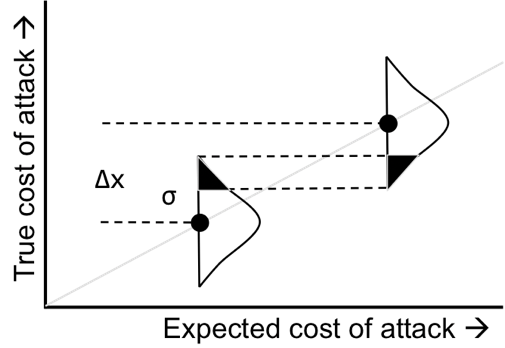
  
(a) Attack profile with certainty (σ = 0). (b) Attack profile with uncertainty (σ > 0).

图2：确定性（左）和不确定性（右）下的攻击概况。

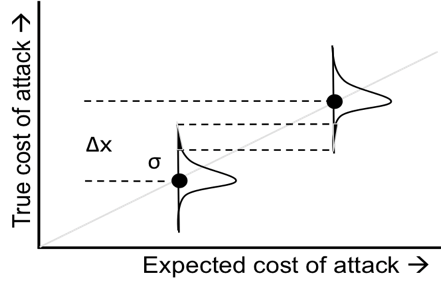
该模型阐明了安全性投资不足可能是一种理性行动，因为：a）可以进行无效投资; b）攻击者利用不同威胁的相对能力存在不确定性; c）成功的攻击不是灾难性的; d）升级防御结构的沉没成本相对较小[6]。该模型适用于网络犯罪和支付安全中的网络钓鱼，随后的论文[33]研究了使用渗透测试作为降低不确定性的手段。

IWL试图模拟对一组资产A的保护，防御者从中获得每段时间的回报率r。受攻击威胁的N个可能组件（如攻击向量）会危及这一点，每个组件都有相应的攻击成本。在IWL中，每个连续的威胁（1，……，n）比先前的威胁增加∆x的成本，形成攻击梯度。在最初的IWL中，这被设置为∆x=1，指定每个防御者投资都会线性增加攻击者的成本。

假定攻击者（x）对n的真实顺序是已知的这些信息对于必须形成可能威胁的预期排序的防御者是未知的（例如通过BSIMM的智能域中指定的攻击建模[17] ）。防守者的排序用x表示（其中x1≤xi≤xn）。如果防御者的攻击模型是完美的，他们将生成与攻击者的个人资料相匹配的攻击配置文件;这是图2a中描述的情况。在这种情况下，我们可以说防御者（点）对攻击（环）具有绝对的确定性。但是，更有可能的是，防御者的估计与攻击者的个人资料不完全一致，导致排序错误。这种情况如图2b所示，其中防御者的不确定性导致不正确的排序（即威胁3和4），导致防守者的投资错位。 IWL将防御者未对准威胁和防御的可能性视为σ，指明防御者的不确定程度。



（a）防御者攻击命令具有很高的不确定性，可能导致攻击向量的错误或顺序。



（b）防御者攻击顺序具有低不确定性，减少了不正确的战术矢量顺序的可能性。

图3：建立防御者攻击命令时不确定性σ与攻击梯度Δx之间的关系。

防御者对IWL的投资在离散时间t =（1，...，tmax）上进行。 在每个t处，防御者形成由向量dt表示的防御配置。dt∈{0,1} n的元素di表示实施了对第i个威胁的防御（di = 1）或不实现（di = 0）。 这些防御的总和用k表示。

（1）每轮保护。

后者用于做出投资决策的知识遵循[6]，第i个威胁的预期成本表示

=+(i-1) (1)

未知的真实成本被建模为高斯随机变量N，平均值为xi，标准差为σ/Δx（删除值为xi>0）：

xi=sup(0,χi) χi ∼ (, σ/∆x ) (2)

σ在IWL中的作用如图3所示：σ值越大，分布越广，增加了防御者排序不正确的可能性，导致安全投资错位（图3a）。然而，在较低的不确定性（即较小的标准偏差）下，该概率降低（图3b）。

攻击梯度的值（Δx）也有助于整体安全投资决策：Δx越大，给定σ的预期威胁成本的重叠越少。这在图2和图3中描绘为实际攻击成本相对于防御者期望分布的线的斜率。因此，这些参数对安全投资的影响最好被认为是比率σ/Δx（表示对于给定量的不确定性，攻击者成本的增加）。我们将这种区分作为[6]中阐述的扩展，因为原始IWL使用了Δx的单位成本来呈现这种区别（σ/Δx=σ/ 1 =σ）。

模型从防御者在t = 0时指定初始k开始。在每个后续轮次（t = 1，...，tmax）中，攻击者可以选择利用dt未涵盖的真实成本最低的组件。从资产a中掠夺z - 但前提是利益超过了攻击成本。面对利用下一个最薄弱环节的攻击者的经济可行性的不确定性（σ），防御者可以执行以下行动。

•忽略攻击并接受潜在的损失。该模型不考虑公共成本，因此当损失低于安全投资需求时，这种私人损失可被视为理性。

•当防御者的位置由于成本和不确定性而变得不可行时，从企业中撤资。

•指定新的防御配置（dt），因为最弱的链接会显示真实排序的每个部分。

然后，在攻击者决定的每个步骤中重复该过程。当低于预留攻击成本的剩余向量受到保护时，达到安全状态。

此外，IWL还包括我们的模型尚未考虑的一个方面，但值得注意的是：当防守者选择在给定回合中改变防守配置dt时产生的沉没成本（λ）。虽然为了简化我们的论述而省略，但我们对SSE流程投资的代表与IWL在部署时的这一方面没有什么不同，并且在所提出的模型中将其包含在部署之后的工作。

我们的模型关注的是IWL等模型所解决的脆弱性集合的起源，以及防御者对此集合的不确定性。这是通过应用SSE来实现的，在提高软件本身的初始安全条件（由攻击梯度Δx表示）的同时降低了防御者的σ。我们在系统集成和部署之前，即在SDLC的采集/开发阶段，考虑缺陷和错误识别的成本和价值。为了做到这一点，我们使用SSE模型作为防御者行动来补充IWL，以极大地消除了潜在的漏洞（与后期阶段的成本相比）。借鉴软件工程的经验，我们想问如何在整个系统的生命周期中最小化这些成本。