doi:10.11959/j.issn.1000-436x.2016166

面向 SaaS 云平台的安全漏洞评分方法研究

李舟，唐聪，胡建斌，陈钟
（北京大学信息科学技术学院，北京 100871）

摘 要：对不同的第三方提供的云服务进行漏洞评分是一项充满挑战的任务。针对一些基于云平台的重要因素，例如业务环境（业务间的依赖关系等），提出了一种新的安全框架 VScorer，用于对基于不同需求的云服务进行漏洞评分。通过对 VScorer 输入具体的业务场景和安全需求，云服务商可以在满足安全需求的基础上获得一个漏洞排名。根据漏洞排名列表，云服务提供商可以修补最关键的漏洞。在此基础上开发了 VScorer 的原型，并且证实它比现有最具有代表性的安全漏洞评分系统 CVSS 表现得更为出色。
关键词：SaaS；云服务；漏洞评分系统；CVSS
中图分类号：TP393 **文献标识码**：A

Vulnerabilities scoring approach for cloud SaaS

LI Zhou, TANG Cong, HU Jian-bin, CHEN Zhong
(School of EECS, Peking University, Beijing 100871, China)

Abstract: There are full of challenges to score vulnerabilities of cloud services developed by different third-party providers. Although there have been a few systems for scoring vulnerabilities (e. g., CVSS) of many existing software, most of them are unable to be leveraged to score vulnerabilities in cloud services, because they fail to consider some important factors located in the clouds such as business context (i. e ., dependency relationships between services). VScorer, a novel security frame work to score vulnerabilities in various cloud services were presented based on different given requirements. By inputting concrete business context and security requirement into VScorer, cloud provider can get a ranking list of vulnerabilities in the business based on the given security requirement. Following the ranking list, cloud provider was able to patch the most critical vulnerabilities first. A prototype was developed and VScorer can be demonstrazed to work better than current representative vulnerability scoring system CVSS.
Key words: SaaS, cloud service, vulnerability scoring system, CVSS

1 引言

云计算越来越受欢迎，它作为大数据的下一代基础设施，应用于许多应用软件中。此外，云计算提供了一种新的商业模式，软件服务和核心计算应要求外包给第三方平台，如苹果的 iCloud 和谷歌的应用平台。最近，Ristenpart 等[1]指出这项新的商业模式也引入了一系列的风险。因为云平台中的软件和应用服务来自不同的服务商，在这些服务和软件中不可避免地存在许多漏洞。不幸的是，以往的研究[2,3]表明，现有的漏洞发现和风险管理技术不适用于评估和处理业务环境的情况。如图 1 所示，定义业务环境为某一项业务，业务间的依赖关系如箭头所示。另外，v_i 用于表示不同服务的漏洞，它呈现不同服务间具体依赖关系图表。综上所述，很有必要研究如何对云服务漏洞进行评级和打分。

事实上，漏洞发现一直都是安全服务行业的重要一环，每年有成千上万的新的服务漏洞被检测并且发布出来[4]，同时投入大量努力去解决这些问题。然而，目前已有的大部分解决方法都是通过打补丁和其他减缓策略，这会耗费大量的人力。

收稿日期：2016-06-17；**修回日期**：2016-08-01
基金项目：国家自然科学基金资助项目（No.61272519, No.61170297, No.61572080, No.61472258）
Foundation Item: The National Natural Science Foundation of China（No.61272519, No.61170297, No.61572080, No.61472258）

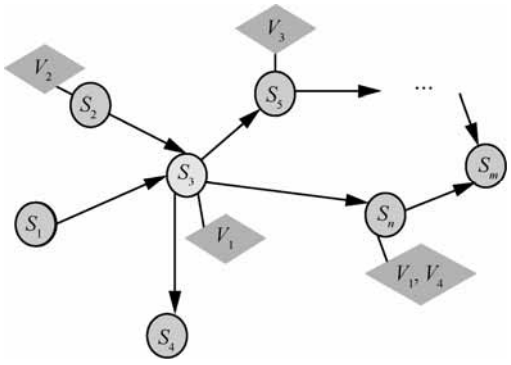

图 1　云服务中业务场景的服务间相关性

因此，目前一些研究者致力于提出解决方案[3]，以期能基于威胁等级来打分，让开发者可以先处理这些比较严重的漏洞。换言之，组织者有充足的人力处理每个发现的漏洞（比如给每个漏洞打补丁），这个方法可能影响到这些公司。因此，云服务开发者应该能够合理分配和规划工作，使最严重的一些漏洞可以优先定位和解决。基于安全方面的考虑这也是很重要的，如果业务中的一些关键服务的某些漏洞被攻击者利用，重要业务就会遭受影响。如图 1 所示，由客户端来执行的某项具体业务环境，对手可以很轻易地利用和攻击服务器 S_3 的漏洞来使整个业务瘫痪，而不是攻击其他服务器的漏洞（如 S_2 和 S_5）。

现有一些系统设计是基于漏洞打分策略[3,5~7]。然而，这些系统存在各种各样的问题，致使无法应用于现在的云服务中：1）未成功考虑业务环境，目前打分系统不仅对于云服务的精确打分漏洞的要求来说过于简单，而且并未成功考虑业务环境，比如服务器间（如软件）的依赖关系，这样并不能够用来在云服务中进行给漏洞评分，目前的漏洞打分系统过于局限，针对漏洞不能提供较强的预测能力[3]；2）未能区分安全需求，目前的漏洞打分系统（比如 CVSS）未能基于不同的安全需求来区分相同漏洞的威胁等级，例如，对于某一业务，某个漏洞可能对于它的可用性来说非常关键，但是对于它的健全性来说并不如此，因此，缺乏安全需求的考虑，致使在漏洞方面以前的系统的可预测能力是难以使人信服的。

为了使云服务更加让人信赖，在云服务中根据具体的安全需求来评估和评级漏洞，本文提出了一种新型架构 VScorer，可以应用于目前云服务基础架构，并且在云服务中根据特定的安全需求进行漏洞打分。具体而言，对于某个业务环境，通过向 VScorer 系统中输入业务环境（比如服务器间的依赖关系）以及安全需求，云服务开发者（比如云服务公司的安全开发者）能够获得业务中存在的漏洞评级列表。这份漏洞评级列表不仅能让云服务开发者明确哪些漏洞应该优先处理，而且能让云服务客户端理解云服务是否处于危险中。

实际上，VScorer 针对具体的业务环境对某特定的漏洞 v 进行打分，依赖于以下 3 个因素：1) 对于特定安全需求（或者成为安全特性）中 v 的威胁等级；2) 包含漏洞 v 的服务的重要性；3) 漏洞 v 的可利用性。定义服务器 S_i 中漏洞 v 的可利用性为可以从 S_i 中利用的可能性。对于某些漏洞，不同的服务中可利用性是不同的。在实际的应用中，安全开发者按照以下 3 个步骤来执行 VScorer。

1) 针对特定的业务构建或者获取服务器间的依赖关系图表，并且将相关性和具体的安全需求输入 VScorer。一些软件和应用[8,9]可以自动生成产生服务器间的相关图。

2) VScorer 基于之前提过的 3 个因素来计算每个漏洞的得分，并且给所有漏洞来评级，产生基于特定安全需求的漏洞评级列表。

3) 根据评级列表，云服务的开发者通过某些有效方法（比如给漏洞打包）来处理这些漏洞。

2　场景和系统框架

VScorer 设计为可以在云服务中用于评分漏洞的安全框架，适用于：1) 不同环境的很多业务，每个业务由可能包含漏洞的服务组成；2) 场景中有 3 类身份，即普通用户、云开发者和攻击者。相对来说，普通用户目标是执行他们想要的业务，他们可称为云服务的客户端；云服务开发者的目标是确保整个云服务的安全；攻击者利用漏洞来攻击不同的服务。

如图 2 所示，服务器的相关图（如业务环境）和安全需求输入 VScorer。一般而言，VScorer 的主要任务是在业务环境中进行漏洞打分，并且根据输入的安全需求输出漏洞的评级列表，云服务开发者以此来对最重要的漏洞优先进行处理。服务器 S_1~S_n 输入到生成器来获取服务间的相关图。相关图表和具体安全需求输入到基础架构中，VScorer 能够生成根据漏洞得到的评级列表。对于某个漏洞 v，VScorer 关于 v 的评分算法主要基于 3 个因素：包含 v 的服务器的重要性、安全需求以及不同服务 v 的可利用性。

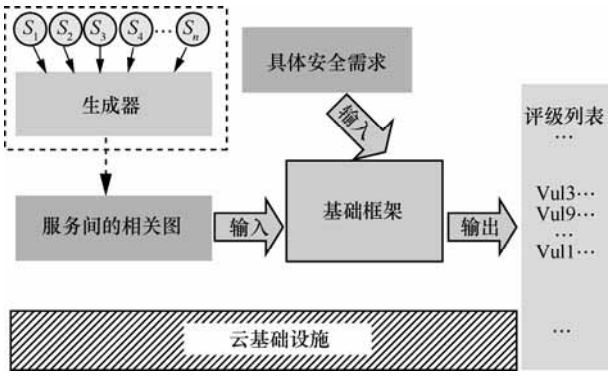

图 2　VScorer 系统框架

3　Vscorer 的设计

3.1　目标

VScorer 的设计主要回答以下 2 个问题。

1) 假定某个安全需求，具体业务环境中哪个漏洞是最重要的（比如最危险的）。

2) 怎样合理并且精确地为这些漏洞打分。

下面来讨论用于设计 VScorer 系统漏洞打分算法的因子。

3.2　讨论

后续描述都会设计到给定的漏洞集 V'。为了合理地计算 V'中所有漏洞的分数，有 2 个影响到 V'中每个元素（v_i）打分的事情：1) 计算中所考虑到因素（或者部分）；2) 获取这些因素的方法。下面将分开讨论这 2 个影响因素。

计算每个漏洞得分时，需考虑以下 3 个要素。

1) 包含 v_i的服务器的重要性。因为所面对场景是云服务，这些服务器存在很多依赖关系（比如业务环境），服务的重要程度有必要考虑到。以前的策略（比如 CVSS[5]和 USCERT[10]）并未考虑到业务环境。例如，作为具有代表性的漏洞打分系统，CVSS 的“漏洞基础得分”按照影响因子 I 和可利用度 E 要素来表示的

$$BaseScore = 1.176\left(\frac{3I}{5}+\frac{2E}{5}-\frac{3}{2}\right) \qquad (1)$$

影响因子(I)是由 CVSS 的开发者或者专家提供的。很明显，CVSS 的算法不能应用于云服务基础架构中，因为等式的 2 个要素与云服务的业务场景并无关联。

2) 按照给定安全需求的 v_i的威胁等级。几乎所有目前的工作都未考虑不同安全需求中漏洞的威胁等级这个因素。为了更加合理可信地对 v_i进行打分，需注意到不同安全特性下对 v_i的威胁等级评估的必要性。比如，对于某个特定服务，某个漏洞对这个服务的可用性非常关键，但是对于健全性远远没那么重要。因此，对于某个漏洞，有必要考虑不同安全特性下的威胁等级。将这个因素设计为一个函数，它的输入是具体的威胁 v_i、安全需求 p_j，返回的值是关于安全需求 p_j 的 v_i 的威胁等级。

3) 不同服务中 v_i 的可利用性。如上面所提到的，漏洞的可利用性应该考虑到。实际上，以往的系统，如 CVSS 和 USCERT 也考虑到这个因素。考虑到 CVSS 已经发布很多可靠的数据和可利用性的信息，因此，本文的框架直接从 CVSS 系统中提取可利用性的信息。事实上，以往的工作[3]也直接从 CVSS 中使用了这部分数据。

总而言之，以往的系统（如 CVSS 和 USCERT）并未考虑到业务环境，它们并未成功合理地对云服务进行漏洞评分。更进一步说，以往所有致力于漏洞评分的研究只考虑了 2 个因素：影响力（比如可利用性的结果的重要程度）以及可利用性（比如漏洞利用的难易程度），这些考虑并不充分。

实际上，VScorer 能够从一些已有系统（如 CVSS）的信息获取这 3 个因子。这是 VScorer 主要贡献之一，因为，以往的工作需要一些专家学者的参与。自动获取这 3 个因素的方法如下所示。

包含漏洞的服务重要性评估计算参考了如 PageRank 和 AssetRank[11]的重要程度计算算法。由于云服务中存在特殊的特征（比如云服务间存在直接的依赖关系），重要性算法可以认为是一项给云平台场景中的不同服务进行评级的有效方法。

为了获取漏洞的可利用性，VScorer 直接利用了 CVSS 或者其他漏洞评分系统的可利用性评分。以往的研究[3]指出，CVSS 的可利用性可直接借用，但是 CVSS 其他的要素（如漏洞的影响程度）应该由某些安全专家提供。因此，可利用性度量来自 CVSS，实际上并未违反设计自动化的原则。另一方面，CVSS 的可利用性支持本文的理念，以往研究中的可借鉴之处是应该直接借用的。

为了获取给定安全需求下的某些漏洞的威胁等级，VScorer 使用了两大在线漏洞数据库提供的信息：OSVDB[12]和 CVE[13]。例如，OSVDB 包含超过 57 000 个漏洞的报告，并且能够提供这些漏洞在不同安全特性下的威胁等级信息。总之，可以基

于自动化方法和现有系统获取 3 个因子。这不仅节约了人力成本，也充分利用了以往研究成果。

3.3 漏洞评分

基于以上讨论和描述，图 3 给出了 VScorer 结构的详细设计。接收到业务环境和安全需求的数据后，VScorer 结合从 3 个组成部分（重要程度算法、VE_DB 和 FS_DB）中提取的 3 个因素来计算所有漏洞的得分。VE_DB 和 FS_DB 是分别存储不同安全需求下漏洞可利用性和威胁等级的数据库。最后一步，此架构生成基于漏洞威胁等级的评级列表。

由于 VScorer 是基于可扩展的架构，图 3 中并未标示出系统要素的详细算法。接下来，将给出计算具体安全需求下给定漏洞 v 的得分的具体算法。

$$R(v)=\alpha\sum_{i=1}^{|S_v|}(E(v,s_i)Rank(s_i))+\beta F(v,P)\sum_{i=1}^{|S_v|}Rank(s_i) \tag{2}$$

其中，S_v 是包含漏洞 v 的服务器集，s_i 是 S_v 集的元素，如漏洞 v 的服务。$E(v, s_i)$是函数返回的可利用性，如漏洞 v 能被服务 s_i 利用的可能性。$Rank(s_i)$是服务 s_i 的重要性，是由重要性算法（如 PageRank 和 AssetRank）计算出的服务器得分。P 是用于澄清语义的安全需求。$F(v,P)$是此函数用于计算某特定安全特性 P 的漏洞 v 的威胁等级，如返回 0~5 的某个数值，以对关于 P 的 v 的威胁等级来进行评级。α 和 β 是基于[0,1]的可调参数。

众所周知，对漏洞进行评级和打分是非常具有挑战性的[2]。接下来讨论式(2)对于漏洞评分的合理性。

1) 因为漏洞的严重性直接相关于包含它们的服务的重要性，不同服务的重要性明显是漏洞评级的一个关键因素。对于某给定漏洞 v_i，包含 v_i 的服务的重要程度应该用于衡量 v_i 的威胁等级和可利用性，以控制威胁等级和可利用性的波动范围。

2) α 和 β 是控制式(2)中 2 部分特性的 2 个关键可调参数。实际上，它们具体的数值应该基于实际需求。例如，实际中如果漏洞的可利用性比它的威胁等级更重要时，α 的值应该设置更大一些。并且，当分别设置 α=1、β=0.5 和 α=0.5、β=0.5 时，本文通过公式得到类似的评级列表。

在一些实际场景中，服务重要程度的计算可能需要考虑到服务间相关性的度量，如有区别地看待相关性，这将包含业务场景进程而不是目前的一个。对于这种情况，VScorer 允许管理者根据具体需求来调整 $Rank(s_i)$的计算，因此，$Rank(s_i)$实际上是能够适用于不同场景的通用的重要程度计算算法。另外，如第 2 部分所提到的，尽管目前 VScorer 根据 PageRank 算法来计算重要程度；然而，更多特殊的关系，如服务器间需要考虑“或”和“与”，AssetRank[11]应该替代 PageRank 来作为重要程度算法；同理，如果目标业务场景的某些服务已经瘫痪，应该借助 TrustRank[14]来为每个服务进行打分和区分。因为目前 CVSS 和 USCERT 漏洞打分系统不能做到这些，VScorer 的通用性和适应性要比已存在的研究成果更加优秀。

图 4 所示的例子可以更清晰地理解式(2)。示例中有 9 个服务器（S_1~S_9），其中有些有漏洞（v_1~v_3）。云管理者 Alice 试图评估这 3 个漏洞，它将服务器的依赖关系输入 VScorer 系统；同时，它评估的安全需求设计为“integrity”。为了简化计算，假设不同服务器中漏洞的可利用性是 1.0，即 $E(v_1, s_1) = E(v_1, s_4) = E(v_2, s_4) = E(v_2, s_7) = E(v_3, s_6) = E(v_3, s_9)$。对于不同的漏洞威胁级别，安全的需求是鉴权，即 P=“integrity”），假设 $F(v_1, P)$=1，$F(v_2, P)$=2，$F(v_3, P)$=3。VScorer 首先通过 pageRank 算法得到所有服务的 $Rank(s_i)$，从

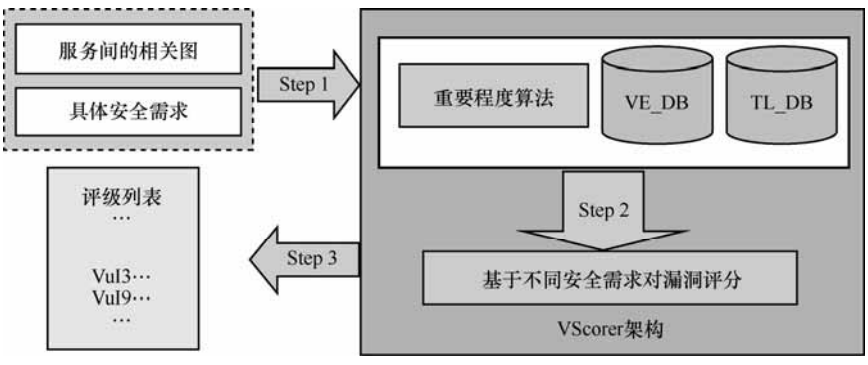

图 3 VScorer 的具体方案

而获得 $Rank(s_1)$=0.15, $Rank(s_2)$=0.21, $Rank(s_3)$=0.21, $Rank(s_4)$=0.24, $Rank(s_5)$=0.24, $Rank(s_6)$=0.33, $Rank(s_7)$= 0.35, $Rank(s_8)$= 0.5, $Rank(s_9)$=1.01。然后 VScorer 可以通过式(2)计算出 v_1、v_2 和 v_3 的 Alice 分数。

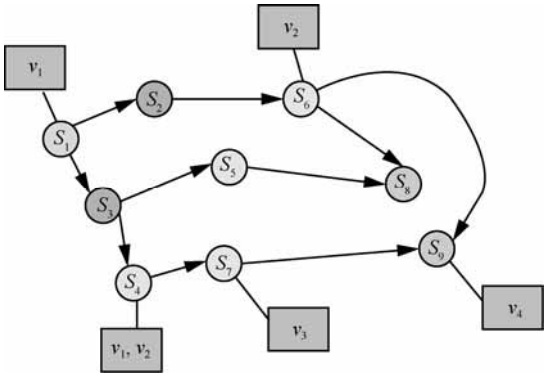

图 4　通过式(2)对漏洞 v_1、v_2 评分

$$R(v_1)=\alpha\sum_{i=1}^{|S_{v1}|}(E(v_1,s_i)Rank(s_i))+\beta F(v_1,P)\sum_{i=1}^{|S_{v1}|}Rank(s_i)$$
$$=\alpha(0.15+0.24)+\beta\cdot 1\cdot(0.15+0.24)$$
$$=0.39(\alpha+\beta) \quad (3)$$

α 和 β 如果赋值为 0.5，那么漏洞排名是：$R(v_3)$=2.68, $R(v_2)$ =0.885 和 $R(v_1)$=0.39。

3.4　另一种选择

虽然已经证明了式(2)可以在云端对漏洞评分，本文还是希望呈现更多能被选择的算法来对漏洞进行打分。因此，提出了另一个公式用来对给定的漏洞 v 打分，如式(4)所示。

$$R(v)=\alpha\sum_{i=1}^{|S_v|}E(v,s_i)Rank(s_i)F(v,P) \quad (4)$$

式(4)中所有符号的解释在式(2)中已经给出。

研究表明，符合条件的风险是一个棘手的问题[2]。因此没人能证明该算法评估的风险（例如漏洞）是最准确的。最好有多种不同的选择（即不同的公式）对漏洞进行打分，通过实验和模拟不同的情况来评估哪些是最合适的。事实上，此功能体现了本文框架的灵活性和可用性，因为 VScorer 能够通过部署不同的漏洞评估机制来满足各种实际的需求。

3.5　实际问题

除了上述讨论，在现实中的应用，还有如下 2 种需要考虑的实际问题。

1) 有一些服务或者软件在漏洞排名前已经被泄露。

2) 由于隐私方面的一些原因，某公司不希望公开服务的依赖关系。

要回答上述实际问题，需要通过引入一些新的机制来加强 VScorer。

1) 对于第 1 个实际问题，本文引入 TrustRank[14] 来替代 PageRank 计算各服务的重要性和信任关系，以此来发现攻击者的服务妥协；另一方面，TrustRank 和 PageRank 在计算服务的重要性方面具有相同的功能。因此，引入 TrustRank 对服务进行打分不会影响对漏洞的打分。

2) 对于第 2 个实际问题，可以要求公司自己对服务进行排名。请注意，这个排名列表仅包含由该公司提供的服务顺序，而不是服务的重要性。在现实中，要求一个公司给出他们服务的顺序（基于重要性）应该不是问题。这是因为对每个服务评级对一个公司来说很难，但是他们必须明白哪些服务是最重要的，相对重要和不重要的。

实际上，附加采纳上述 2 种解决方案不是什么问题。VScorer 的设计使添加其他组件或模块是非常方便的。

4　评估

本节的主要目的是评估 VScorer 的性能，建立 VScorer 原型系统；然后通过 3 个案例，基于真正的业务环境和模拟漏洞信息来验证 VScorer 的有效性。

为了评估 VScorer，本文提出了 3 个基于真实业务流程的案例，用 CVSS 对比 2 个公式（即式(2)和式(4)），从而表明 VScorer 的有效性。每个案例的研究如下。

1) 输入具体的包括安全需求和业务流程的实验数据进入 VScorer 模型，从而获得对给定的安全需求的漏洞排名。分别通过式(2)和式(4)生成 2 个排名列表；因此，实际上是由 VScorer 获得 2 个排名。

2) 通过 CVSS 生成另一个排行榜，比较 CVSS 的结果和之前通过 VScorer 的 2 个公式生成的排行榜。使用 CVSS 来比较 VScorer 的原因将在后面给出。

3) 比较并评估哪一个排行榜是最好的，通过模拟评估漏洞排行榜。之后，分析和讨论得到的结果。

目前，已经在实际使用中的几个漏洞评分系统，CVSS 已经逐渐成为公共的标准，因此选择 CVSS 和 VScorer 比较。此外，现有的研究[3]也指出 CVSS 是目前具有代表性的漏洞评分系统。接下来，讨论如何比较和评估不同系统的排行榜，3 个研究案例将给出。

4.1 如何比较不同的排行榜

假定每个服务运行在给定的业务流程中是不同的，因此，当运行一个具体的任务在那个业务中时，可以通过在任务运行过程中所有服务的运行时间的总和获取该任务的运行时间。比如，在图 5 中，如果得到 VCL 接入服务的运行时间为 0.3，VCL 管理服务的时间为 0.5 以及应用服务 3 的运行时间为 0.4，所以任务的运行时间为 0.3+0.5+0.4=1.2。此外，如果某个漏洞被攻破，执行这些有漏洞的服务将花费更多的时间，即可以认为泄露的漏洞延迟了有漏洞任务的运行时间。在模拟中，延迟时间为 0.2，例如在图 5 中，如果 VCL 访问服务有一个泄露漏洞，然后它由 VCL 接入服务任务的运行，VCL 管理服务和应用服务 3 的运行时间应该为 0.3+0.5+0.4+0.2=1.4。

描述如何计算性能指标后，运行模拟器的详细步骤如下。

1) 根据给定的业务流程产生的模拟环境（包括服务、依赖和漏洞）。

2) 在给定的业务流程中分别泄露每个漏洞，并计算每个泄露的漏洞的性能值。特别地，对于每个漏洞，随机在指定的业务中执行不同的任务 1 000 次，然后计算运行 1 000 次的平均时间作为该漏洞的性能指标。

在获取不同泄露漏洞的性能指标之后，可以评估哪一个排行榜最优。显而易见的是性能指标应该是最高的（即延迟时间最高），如果最关键的漏洞被攻破。也就是说，如果 v_3 是最关键的漏洞，模拟器的性能指标应该是最高的。因此，在得到所有性能指标的排行榜之后（从高到低），比较 2 个排行榜之间的匹配程度，即漏洞排名和性能指标排名（性能指标排序由高到低）。2 个排行榜更匹配的就是更准确的漏洞排行榜。下面将提出 3 个研究案例来评估本文方法。

4.2 案例 1：基于 VCL 数据集的评估

在这个案例中，使用一个来自 VCL 的真实业务流程来评估 VScorer，业务流程的信息展示在图 5 中，菱形表示漏洞，其他表示 VCL 中的服务。漏洞的可利用性和威胁级别（对于给定的安全需求）展示如表 1 所示，假设不同的服务中相同漏洞的可利用性是一样的。威胁等级是根据给定的安全性要求在 0~5 中取值。请注意，只能从 VCL 中提取业务流程，但不是所有的漏洞。由于隐私问题，本文通过仿真生成漏洞。类似地，案例 2 和案例 3 利用真实的业务流程和漏洞进行了模拟。

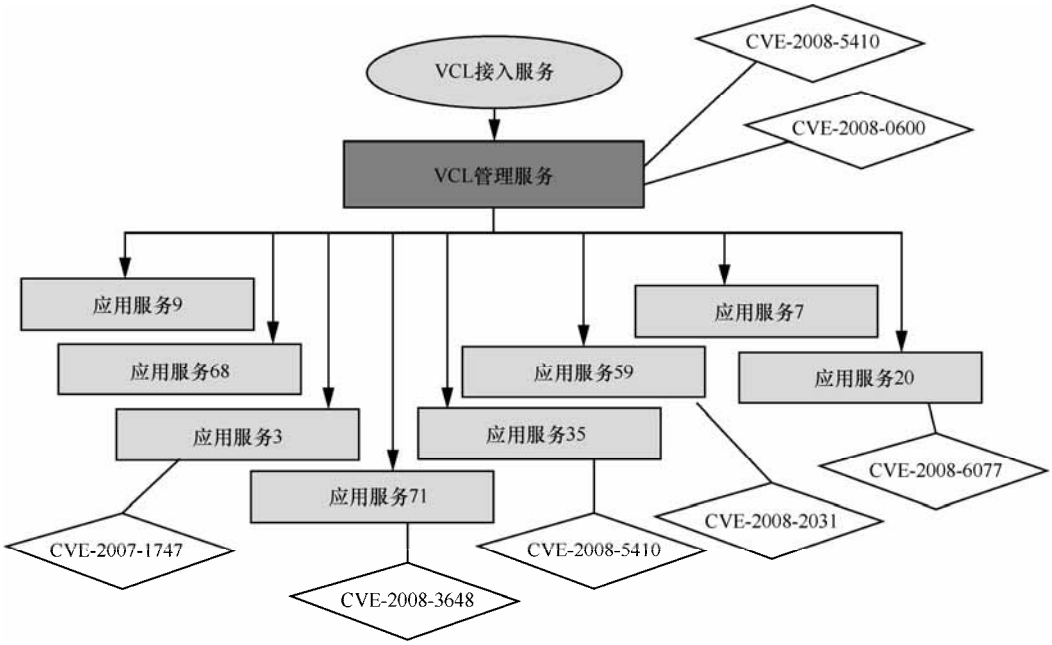

图 5 VCL 数据集的具体业务流程

表 1　案例 1 漏洞的可利用性和威胁级别

漏洞	可利用性	威胁级别
CVE-2007-1747	0.801	2.0
CVE-2008-3648	0.45	2.0
CVE-2006-6077	0.304 5	3.0
CVE-2008-5410	0.702	2.0
CVE-2008-0231	0.34	1.0
CVE-2008-0600	0.54	3.0

基于 VScorer 的 2 个公式，即式(2)和式(4)，得到了 2 个对于给定安全需求漏洞的排行榜；同时，使用 CVSS 产生了另一个排行榜。表 2 展示了这 3 个排行榜。相对于 α 和 β 的值，设置 α=1.0，β=0.5。实际上，也可以为它们设置不同的值（如 $\alpha = 0.5$，$\beta = 0.5$），并且得到一个相同的结果；因此 α 和 β 的值实际上不能显著影响结果。

表 2　　案例 1 VCL 的漏洞排名

式(2)	式(4)	CVSS
CVE-2008-5410	CVE-2008-5410	CVE-2008-0600
CVE-2008-0600	CVE-2008-0600	CVE-2006-6077
CVE-2007-1747	CVE-2007-1747	CVE-2007-1747
CVE-2008-3648	CVE-2006-6077	CVE-2008-5410
CVE-2006-6077	CVE-2008-3648	CVE-2008-3648
CVE-2008-0231	CVE-2008-0231	CVE-2008-0231

评估排行榜并且讨论。要了解 3 个排行榜的准确性，本文的模拟器执行评估 3 种方法的有效性。当漏洞 CVE-2008-5410 被攻破时性能指标最高，根据该指标从高到低，模拟器基于性能指标对漏洞进行的排序为：CVE-2008-5410、CVE-2008-0600、CVE-2007-1747、CVE-2008-3648、CVE-2006-6077 和 CVE-2008-0231。实际上，这个顺序与 VScorer 产生的排行榜正确匹配；另一方面，由于未能考虑给定安全需求的业务环境和威胁级别，CCVS 的排行榜是不准确的。因此，在案例 1 中，实验表明 VScorer 可以工作得比 CVSS 更好。

4.3　案例 2：评估基于 EC2 的业务流程

在此案例中，本文使用一个 EC2[15]的真实业务流程。图 6 所示为服务的具体依赖关系和位于业务流程中的漏洞，菱形表示漏洞，其他表示 EC2 的不同服务。漏洞的可利用性和威胁级别（根据安全要求）如表 3 所示。

表 3　案例 2 漏洞的可利用性和威胁级别

漏洞	可利用性	威胁级别
V-2009-4447	0.7	3.0
V-2009-3508	0.5	2.5
V-2006-6077	0.7	4.0
V-2007-6410	0.6	3.5
V-2008-3631	0.5	2.5

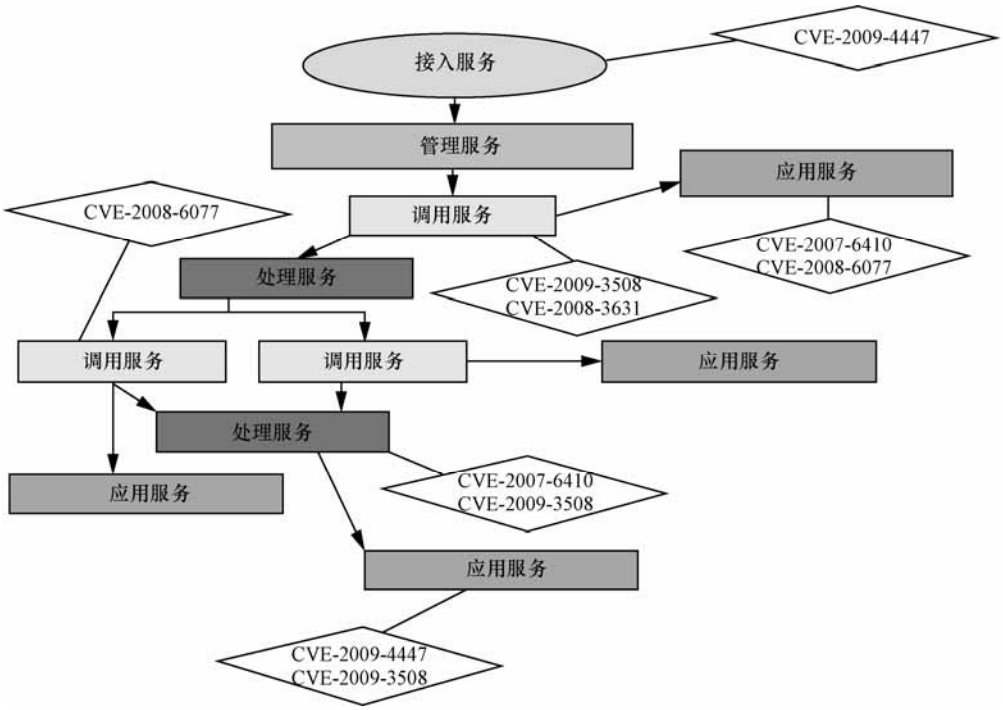

图 6　EC2 数据集的一个具体业务流程

表 4 案例 2 EC2 特定业务流程的漏洞排名

式(2)	式(4)	CVSS
V-2009-3508	V-2006-6077	V-2006-6077
V-2007-6410	V-2009-3508	V-2007-6410
V-2006-6077	V-2007-6410	V-2009-4447
V-2009-4447	V-2009-4447	V-2009-3508
V-2008-3631	V-2008-3631	V-2008-3631

运行模拟器之后，模拟器的结果表明漏洞的正确顺序应该为：V-2009- 3508、V-2007-6410、V-2009-4447、V-2006-6077 和 V-2008-3631。另一方面，如表 4 所示，可得出 2 个结论：1) 式(2)得出的排行榜是最准确的，因为它和模拟器得出的结论最匹配；2) 虽然式(4)得出的结果在案例 1 中是正确的，它不能在本案例中给出一个正确的结果；同时，CVSS 的结果也不准确。实际上，这是因为 2 种方法（VScorer 的式(4)和 CVSS）背后的算法。特别地，CVSS 没有考虑服务间的依赖，而式(4)具有算法方面的薄弱点。因此，我们证明了 VScorer 的式(2)比其他 2 种方法工作得更好。

4.4 案例 3：评价 IBM 的灾难援助理赔服务

案例 3 使用的数据和信息来自于 IBM 的灾难援助理赔服务。如图 7 所示，菱形表示漏洞，其他表示业务中的不同服务，可以看出具体的依赖关系和位于各种服务上的漏洞。漏洞的可利用性和威胁级别（根据安全要求）如表 5 所示。

表 5 案例 3 漏洞的可利用性和威胁级别

漏洞	可利用性	威胁级别
V-2010-4307	0.71	3.0
V-2009-2208	0.53	2.5
V-2010-9337	0.212	2.0
V-2009-9010	0.103	2.5
V-2009-1031	0.503	2.5
V-2009-2675	0.52	2.5

表 6 案例 3 IBM 的灾难援助理赔服务的漏洞排名

式(2)	式(4)	CVSS
V-2009-2208	V-2009-2208	V-2010-4307
V-2009-2675	V-2009-2675	V-2009-1031
V-2010-9337	V-2010-4307	V-2009-2675
V-2009-9010	V-2010-9337	V-2009-2208
V-2010-4307	V-2009-9010	V-2009-9010
V-2009-1031	V-2009-1031	V-2010-9337

评估排行榜并且讨论。运行模拟器之后，可得出 3 个结论：1) 从式(2)得到的排行榜在本案例中对漏洞的排序是正确率最高的，因为它和模拟器得出的结论最匹配；2) 式(4)的结果也是准确的，但是不如式(2)精准。事实上，式(4)在本案例中至少比 CVSS 表现得更好；3) CVSS 的算法在本案例中没有得到一个好的结果，因为 CVSS 的机制不能在考虑业务流程的情况下，取得良好的效果。因此，可以得出 VScorer 的式(2)应该是最准确的，并且在云服务中可以工作得比 CVSS 更好。

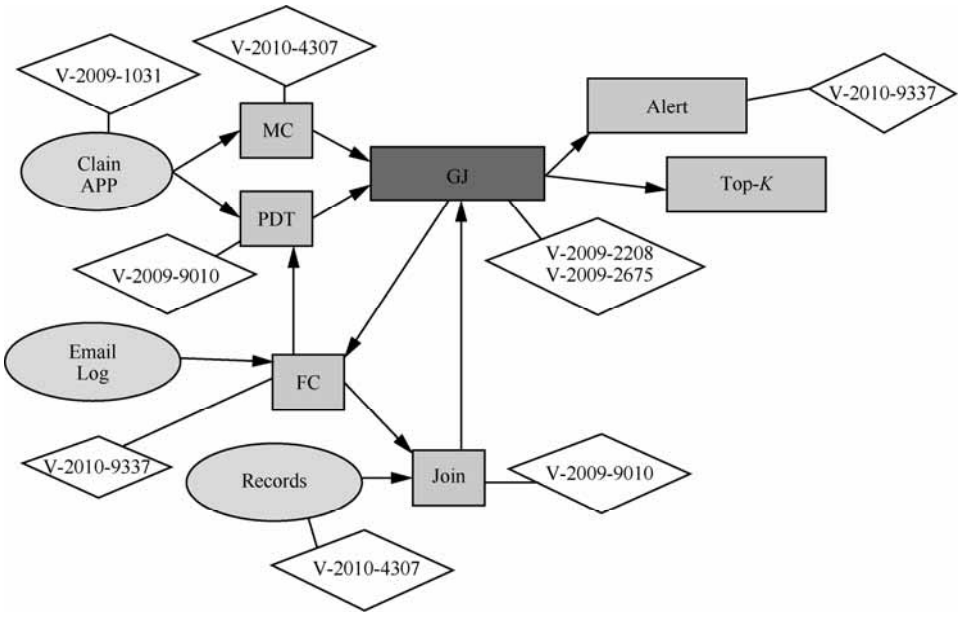

图 7 IBM 的灾难援助理赔服务的一个具体业务流程

3 个案例得到的漏洞评级列表如图 8 所示。

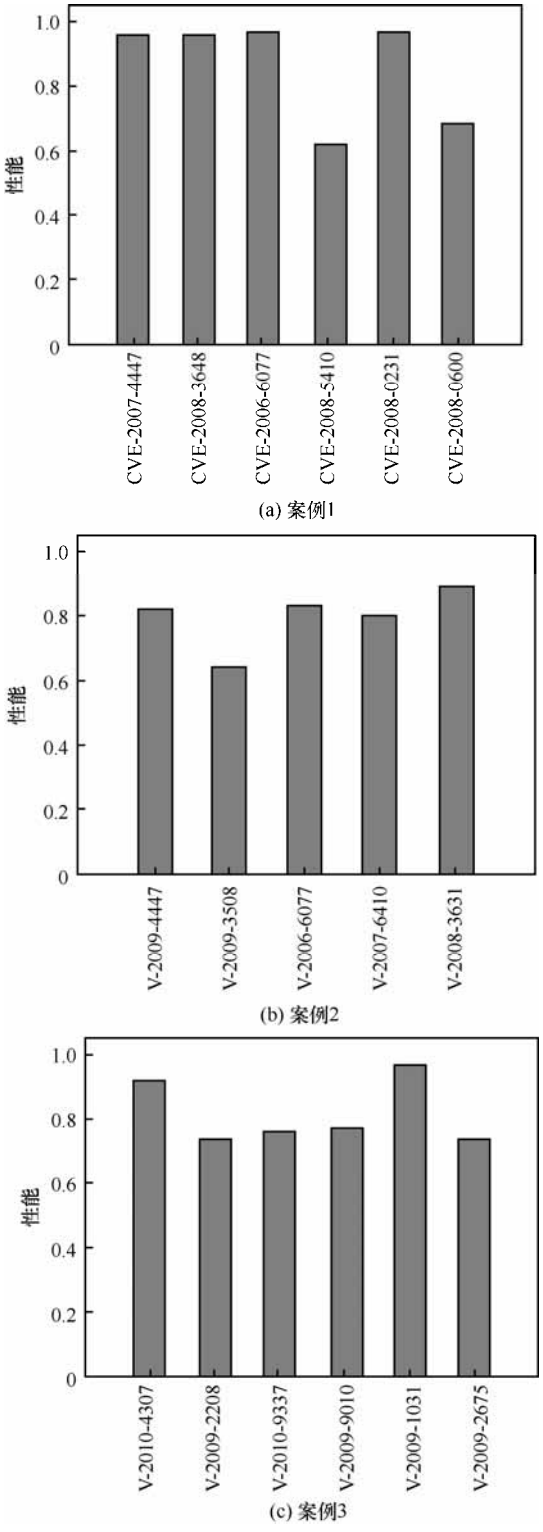

图 8　在 3 个案例中基于仿真器性能得到的漏洞评级列表

为了帮助安全管理员，许多漏洞评分系统已经被提出。几乎所有的人都关注于如何基于 2 个因素对漏洞打分：影响和可开发性。很明显，它们不能被采用到云基础设施中，因为它们没有考虑具体的业务流程上下文这个云服务中的关键因素。特别地，USCERT 产生了一个 0~180 内的量化程度评分，从一系列的定性问题的答案中直接计算[10]。另外，微软的安全公告记录漏洞的严重性（用定性的方法），用来作为 Secunia 的报告。近来，一个新的严重性指标，常见漏洞评估系统（CVSS）[5]，被一些安全专家和研究人员所提出。CVSS 定义了一些独立的指标；然而，根据这项研究[3]，CVSS 只是“基础指标”，被典型的使用于第三方漏洞数据库。事实上，以前进行了一些努力来分析和改进 CVSS[16,17]。与本文的方法相比，CVSS 未能提供其明确考虑到业务流程上下文的机制。相反，为了克服这些困难，VScorer 采用 PageRank 算法，来计算云服务的服务重要程度。实际上，本文进一步补充用于计算云服务重要程度的 PageRank 的算法，加入具体安全需求的考虑。

目前，很多关于漏洞发现的报告已经通过多资源如 CVE[16]和 OSVDB[12]发布出来。特别是 CVE 明确地将某些安全需求下的不同漏洞威胁等级列入考虑之列；然而 CVE 并未为漏洞评分提供合理的机制。而且，大量研究检测到可被修复漏洞的可能性[18]。

5　结束语

本文设计了一种新型架构 VScorer，来为云服务中的漏洞进行评分和评级。与以往研究成果不同，VScorer 不仅考虑到它的内在特性，如可利用性，而且将可利用性的这些服务场景考虑在内，比如在组成的服务器中所扮演的角色，以及服务安全目标的重要性。漏洞的评分和评级结果对于组成的服务而言具有高相关性和价值。通过基于实际中服务场景进程的实验，对 VScorer 的有效性进行了评估，并且将 VScorer 和 CVSS 进行了比较，结果表明在云服务的漏洞评分场景中，VScorer 比现有的研究成果表现更为出色。

参考文献：

[1] RISTENPART T, TROMER E, SHACHAM H, et al. Hey,you,get off of my cloud: exploring information leakage in third-party compute clouds[C]//ACM Conference on Computer and Communications Security.c2009:199-212.

[2] BELLOVIN S. On the brittleness of software and the infeasibility of security metrics[J]. IEEE Security and Privacy, 2006,4(4): 96.

[3] BOZORGI M, SAUL L, SAVAGE, et al. Beyond heuristics: learning to classify vulnerabilities and predict exploits[C]//ACM Sigkdd International Conference on Knowledge Discovery & Data Mining. ACM, c2010:105-114.

[4] IBM. IBM Internet Security Systems X-Force 2008 Trend and Risk Report[R]. White paper, 2009.

[5] A complete guide to the common vulnerability scoring system[S].

[6] OWASP Top Ten[EB/OL].http://www.owasp.org/,2013.

[7] SANS Top-20 Security Risks[EB/OL]. http:// www. sans. org/ top20, 2009.

[8] CHEN X, ZHANG M, MAO Z, et al. Automating network application dependency discovery: Experiences, limitations, and new solutions[C]// Usenix Symposium on Operating Systems Design & Implementation. c2008:117-130.

[9] ENSEL C. A scalable approach to automated service dependency modeling in heterogeneous environments[C]// IEEE International Enterprise Distributed Object Computing Conference, c2001:128-139.

[10] DOUGHERTY C. Vulnerability metric[EB/OL]. https :// www. securecoding.cert.org/confluence/display/seccode/Vulnerability+Metric, c2008.07.24.

[11] SAWILLA R and OU X. Identifying critical attack assets in dependency attack graphs[C]// European Symposium on Computer Security-esorics. c2008:18-34.

[12] OSVDB. The open source vulnerability database[S].

[13] CVE Editorial Board. Common vulnerabilities and exposures: the standard for information security vulnerability names[S].

[14] GYONGYI Z, GARCIA H, PEDERSEN J. Combating web spam with trustrank[C]//Thirtieth International Conference on Very Large Data Bases. c2010: 576–587.

[15] CHRISTOS T. Software for Cloud[S].

[16] SCARFONE K, MELL P. An analysis of cvss version 2 vulnerability scoring[C]//International Symposium on Empirical Software Engineering & Measurement. c2009: 516 -525.

[17] FRUHWIRTH C, MANNISTO T. Improving cvss-based vulnerability prioritization and response with context information[C]//ESEM, c2009: 535-544.

[18] MOORE D, SHANNON C, CLAFFY K. A case study on the spread and victims of an Internet worm[C]//Internet Measurement Workshop. c2002: 273-284.

作者简介：

李舟（1987-），男，湖北荆州人，北京大学博士生，主要研究方向为信息安全、基于身份的公钥加密。

唐聪（1984-），男，湖南永州人，北京大学博士生，主要研究方向为信息安全、云计算、社交网络。

胡建斌（1971-），男，湖北洪湖人，北京大学副教授，主要研究方向为云计算、物联网、计算机网络安全。

陈钟（1963-），男，江苏徐州人，北京大学教授、博士生导师，主要研究为密码学、计算机网络与信息安全。