自适应的时空多样性联合调度策略设计

仝青[1]，郭云飞[1]，霍树民[1]，王亚文[1]，蔺羽佳[2]，张凯[3]

（1. 信息工程大学信息技术研究所，河南 郑州 450015；2. 中国人民解放军 32066 部队，云南 昆明 652200；
3. 中国人民解放军 31401 部队，山东 日照 276800）

摘 要：为了解决多样性系统在单一多样性策略下存在防御能力、防御代价和服务质量难以兼顾的问题，首先基于调度异构性、执行体安全性和空间多样性度量方法构造不同安全等级下的调度对象选择序列；然后根据对威胁环境的粗粒度评估，综合决策调度时机以及调度对象。通过在云环境下实现时空多样性 Web 服务系统，对所提调度策略进行攻防实验测试，并与已有调度策略进行了对比。结果显示，所提调度策略在可接受的防御代价增长范围内，显著提高了系统的防御能力，同时维持了较高的服务质量。

关键词：多样性；主动防御；自适应性；调度策略；攻防实验

中图分类号：TP393.1

文献标识码：A

DOI: 10.11959/j.issn.1000−436x.2021119

Design of self-adaptive spatio-temporal diversity joint scheduling strategy

TONG Qing[1], GUO Yunfei[1], HUO Shumin[1], WANG Yawen[1], MAN Yujia[2], ZHANG Kai[3]

1. Institute of Information Technology, Information Engineering University, Zhengzhou 450001, China
2. 32066 Army of PLA, Kunming 652200, China
3. 31401 Army of PLA, Rizhao 276800, China

Abstract: To solve the problem that a diversity system is difficult to take defense capability, defense cost and quality of service into account at the same time under a single diversity strategy, firstly, the scheduling object selecting sequences under different security levels were constructed based on the measurement of scheduling heterogeneity, executor security and spatial diversity. Then, according to the coarse-grained evaluation of threat environment, the scheduling time and scheduling object were determined comprehensively. Through the realization of the spatio-temporal diversity Web server system in a cloud environment, the proposed scheduling strategy was tested with attack and defense experiments and compared with the existing scheduling strategies. The results show that the proposed scheduling strategy improves the defense capability significantly and maintains a high quality of service within the acceptable defense cost increasing range.

Keywords: diversity, active defense, self-adaptability, scheduling strategy, attack and defense experiment

1 引言

在网络空间安全中，防御者很难构造一个绝对安全的系统。传统的防御方法主要包括“补”和“防”2 种方式，即打补丁和入侵防御。通过打补丁的方式弥补系统存在的脆弱性，是从根本上防止攻击者对漏洞的利用，然而漏洞具有隐蔽性，防御者难以发掘所有的漏洞并为之打补丁。入侵防御方法通过入侵检测等方式检测攻击的发生并采取相应措施阻止攻击渗透或扩散，但入侵检测存在误报和漏报

收稿日期：2020−12−30；**修回日期**：2021−04−14

基金项目：国家自然科学基金资助项目（No.62072467, No.61521003）；国家重点研发计划基金资助项目（No.2018YFB0804004）

Foundation Items: The National Natural Science Foundation of China (No.62072467, No.61521003), The National Key Research and Development Program of China (No.2018YFB0804004)

的可能且难以有效检测到未知攻击和后门[1]。新型的主动防御技术打破了传统防御技术的被动性，通过改变系统的构造或运行方式，使系统自身内生一定的防御能力[2]。典型的主动防御系统有 SCIT（self cleansing intrusion tolerance）[3]、Talent[4]、MT6D（moving target IPv6 defense）[5]、多变体[6]、拟态防御 Web 服务器系统[7]等。上述系统通过动态改变系统对外呈现的特征增大攻击者发现目标、定位目标的难度，或采用异构化冗余执行降低漏洞的可利用性，最终达到提高系统防御能力的目的。

软硬件多样性是主动防御采用的主要技术之一，依据实现的维度不同，可以分为时间多样性和空间多样性[8]。时间多样性系统在不同的时间段对外呈现不同的安全属性或攻击面，使系统具有动态性。空间多样性系统在同一时间存在冗余异构的执行体或执行过程，将相同的服务请求复制分发到不同的执行体或执行过程，通过表决得到相对正确的服务响应，从而避免单一执行体或执行过程被攻击而导致服务失效。

已有研究对时间多样性系统或空间多样性系统分别进行了安全性、性能等方面的优化与权衡，然而缺少对时间、空间多样性联合机制的设计。另一方面，网络空间中的系统面临的威胁态势时刻在变化，系统应当具备自适应性，而已有的自适应策略主要针对纯粹的时间多样性系统进行设计。针对上述不足，本文提出一种自适应的时空多样性（SASTD, self-adaptive spatio-temporal diversity）联合调度策略，通过评估系统面临的威胁水平，根据时、空多样性各自的优势，适应性地改变系统的多样性配置，使系统在威胁水平较高时，增大防御能力，而在威胁水平降低时，降低防御能力，同时提高服务质量。本文主要贡献介绍如下。

1) 联合了时空多样性共同指导调度决策。结合了 2 种多样性策略的优势，使系统在提高防御能力的同时维持了高服务质量，控制了过多防御代价的引入。

2) 通过适时引入空间多样性降低了自适应策略对入侵检测的依赖，同时使系统具备了一定的防御未知攻击的能力。

2　相关研究

2.1　多样性系统分类

平台层多样性系统的主要构成包括 n 个执行体和一个代理，如图 1 所示。其中代理对外提供服务访问接口，接收用户的输入，按照一定的多样性策略（具体包括分发策略和输出策略）转发请求和响应。用户请求按照分发策略发送到后端执行体进行处理。所有执行体的响应，都要经过输出策略处理后返回至用户。根据多样性不同的实现维度，可以形成不同的多样性策略，大体上可分为时间多样性和空间多样性。

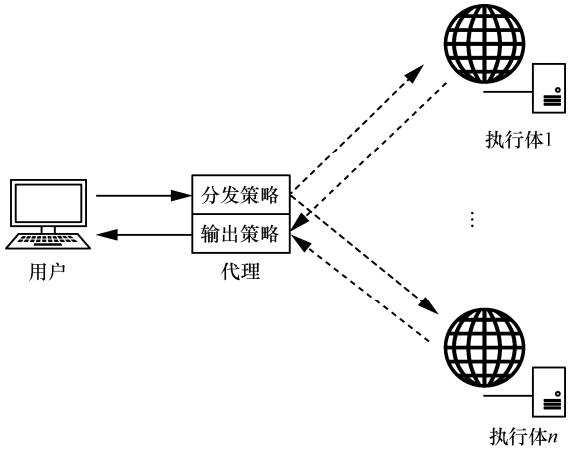


图 1　平台层多样性系统结构

时间多样性系统是指在不同的时间片段内轮换执行体或系统属性的系统，该类系统在不同时间片段内使用不同的执行过程对事务进行处理或对外呈现不同的系统属性。如图 1 所示，代理服务器按照分发策略改变其连接的执行体，使不同时间段下执行体不同，从而增大系统的不确定性。典型的时间多样性系统有移动目标防御（MTD, moving target defense）系统。MTD 系统能够动态地改变攻击面，可以在不同的软件栈层面上实现。例如，在操作系统层实现的地址空间分布随机化（ASLR, address space layout randomization）技术[9]，通过改变程序每次运行时的基地址，使攻击者无法通过先前的破解来定位代码，因为每次程序运行时，加载位置都会随机更改，从而有效抵御控制流攻击对地址的探测；在代码和软件层实现的多版本编程，可以在编译时通过插入空语句等方式，使代码异构化和多样化；在网络层，通过动态变换系统指纹、IP 地址、端口等属性使攻击者难以锁定攻击目标[10]；对于平台层的 MTD 系统，如 TALENT[5]，通过轮换异构执行体上线服务的方式实现动态性。

空间多样性系统是指对一个请求同时采取多个等价的执行过程，并对不同执行过程的执行结果

进行表决以获得最终输出的系统。冗余的执行过程并行存在同一时间片段内，因而称之为空间多样性。如图 1 所示，空间多样性系统的代理服务器按照分发策略将请求进行复制并分发到执行体 $1,2,\cdots,n$，然后收集执行体的响应进行表决，将最终响应返回给用户。典型的入侵容忍系统 SITAR（scalable intrusion-tolerant architecture）[11]采用多组件冗余逐层表决的方法实现入侵容忍。不同的空间多样性系统的主要区别在于表决策略，例如拜占庭表决系统 COCA（cornell online certification authority）[12]在各个节点均进行表决；k-大数表决以 k 个或 k 个以上的一致响应作为最终输出；基于历史信息的加权表决则根据历史表决结果，对出错较多的执行体赋予较低的权值，文献[13]对上述表决策略进行了分析和总结。

不同的多样性技术在防御能力、防御代价以及对系统的服务质量上有不同的影响。

时间多样性系统具有不确定性，攻击者难以定位系统的脆弱点并持续实施攻击，即使攻击成功，也难以长期维持攻击效果。时间多样性将确定性攻击转变为概率性事件，成功概率的大小受系统的动态频率影响。动态频率越高，系统的不确定性越强，攻击成功概率越小，然而频繁的动态变化容易导致系统服务稳定性下降，同时增大调度开销。

对于空间多样性系统而言，攻击者只有在成功利用冗余执行体或执行过程的共模漏洞时，才有可能攻击成功，因而空间多样性系统极大地提高了攻击难度。冗余执行体或执行过程的异构性越大，系统对外呈现的共模漏洞越少，攻击难度越大，这相应地提高了异构性、冗余度的需求，同时对冗余执行结果的表决增大了服务响应时延。

2.2 多样性系统调度策略

自适应的多样性策略具备动态性，因而相关研究中以时间多样性策略居多。文献[14]将时间多样性系统的研究范围划分为调度什么、何时调度以及如何调度 3 个方面。调度什么通常取决于系统的工作层面，如网络层的多样性系统通常动态变换 IP 地址、端口号、主机名等，而平台层通常变换不同的软件版本、虚拟机、容器等[15]。何时调度与如何调度分别解决调度时机和调度对象的选择问题。

自适应的调度时机选择策略多依赖于准确的入侵检测机制。例如，文献[16]描述了一种容错系统的任务调度算法，当检测到正在运行的任务出错时，及时调度异构的其他版本上线替代。该调度算法的目标是维持容错能力，保证任务在规定时间内完成。文献[3]设计了自适应的时间多样性系统 SCIT，该系统通过调度系统的副本上线实现对入侵的容忍，自适应性主要体现在通过评估每个执行体节点的重建时间，给出当前系统中为了保证一定的恢复能力而需要维持在线的节点数量。SCIT 的自适应性是面向系统服务可用性的，而非当前系统所面临的威胁环境。云环境中 SCIT 系统[17]的实现结合了一定的云平台异构性，然而该研究未将异构性进行量化并作为调度对象选择的依据。文献[18]量化了变换代价和攻击代价，并结合历史变换时间间隔，采用更新奖励理论对攻防过程进行分析，以决策下一次变换的时机。这种自适应策略以决策更经济的变换时机为目标，调度时机的选择同时也在一定程度上依赖于入侵检测。文献[19-20]采用博弈模型分析攻防双方的行为和收益，并通过纳什均衡或最优化方法得出防御者需要采取变化的时机，但对入侵检测的依赖性较强且多针对特定的攻击类型，缺乏普遍适用性。

在调度对象的选择方法上，通常同时包含了调度时机选择策略的设计，相较于纯粹的调度时机策略更全面。文献[21]提出了一种以成本为主导的调度策略，该策略同时以时间和事件为调度时机决策因素，通过贝叶斯攻击图评估系统各状态的安全等级，利用竞争马尔可夫决策过程，在调度时采取未来收益最大化的操作进行移动。该研究在调度时机上综合了时间和事件 2 种驱动因素，在调度对象选择上以成本和防御收益为主导，而该策略主要应用了时间多样性，未结合空间多样性以扩展系统能够达到的安全上限。文献[22]提出攻击者具有智能性，其在攻防博弈中具有自适应的特点，因而防御者应根据攻击者行为的变化调整防御策略。该研究基于博弈论和机器学习提出了一种通过预测攻击者行为来采取适应性的防御措施方法，主要采取基于时间多样性的调度，使系统呈现异构性和不可预测性，从而中断攻击者的当前攻击行为并提高下一次攻击发起的难度，该调度策略对入侵检测有较强的依赖性，在应对未知攻击时可能存在漏检。文献[23]提出了网络层攻击面的自适应转换技术，基于对当前网络威胁的感知进行自适应的网络跳变。其中跳变技术在视图距离和跳变周期两方面实现，在网络资

源、可用性、部署效率等条件约束下，通过制定跳变目标实现防御收益最大化。视图距离最大化的目的是提高跳变前后的异构性，而跳变周期的调节改善了跳变的时效性。该研究成果主要应用于网络层，跳变的有效性对威胁感知的准确性有较强的依赖性。另一方面，一旦攻击成功入侵网络节点，后续的跳变对已发生的攻击将无法发挥防御作用。文献[24-26]研究了变换−异构、变换−冗余以及 3 种技术的组合应用，评估了不同情形下组合技术的优劣，并将变换、异构和冗余的组合问题归纳为一个二元线性优化问题，给出了特定限制条件下系统的最优多样性配置。然而，该研究对异构性的度量较模糊。另一方面，冗余的执行体以副本形式存在，用于替换在线执行体，而本文中空间多样性所涉及的冗余是指并行处理相同请求的工作方式。同时，通过最优化问题给出的策略是固定策略，未能形成自适应性。文献[27]通过多样性实现了软件体的自修复，当系统检测到攻击时，针对被利用的漏洞采用遗传算法产生多个多样化的补丁作为软件修复的候选，通过替换当前相关组件，实现软件体的动态性。然而该方法难以保证新产生的补丁不会引入新漏洞，且该研究的调度时机依赖于对攻击类型和漏洞类型的监测和分析。

综上所述，已有研究成果中不乏自适应调度策略的设计，然而主要集中于时间多样性的调度策略设计中。高效的自适应时间多样性策略依赖于准确的入侵检测结果，安全防御能力受到一定的限制。时间多样性与空间多样性在带来安全性增益的同时均对系统存在一定的副作用，可以通过联合时空多样性动态改变对系统安全性、代价和服务质量的影响，实现在不同威胁环境下的优劣互补。然而，尚未有研究对时空多样性的联合调度策略进行设计与评估。

3　策略设计

3.1　面向攻击阶段的防御策略分析

文献[28-29]将攻击阶段划分为目标侦查、信息获取、漏洞挖掘、攻击发起和攻击持续 5 个阶段，可概括为前攻击、攻陷和后攻击 3 个阶段，如图 2 所示。其中，目标侦查阶段，寻找和发现攻击目标，收集目标的基本信息；信息获取阶段，攻击者对目标进行进一步的探测以确定系统架构、操作系统类型等详细信息；漏洞挖掘阶段，攻击者尝试开发目标系统中可利用的漏洞；攻击发起阶段，攻击者通过发送攻击载荷达到攻击目的；攻击持续阶段，攻击者在目标系统上预置木马或后门以便未来再次攻击。

图 2　攻击阶段划分

前攻击阶段。攻击者通常对系统进行信息侦查、漏洞挖掘以及漏洞利用尝试，为了避免被入侵检测系统察觉，上述准备工作往往较隐蔽，漏洞挖掘阶段有时甚至不需要与目标系统交互；由于漏洞挖掘和利用具有一定的难度，前攻击阶段在整个攻击过程中通常耗费时间最长。系统在该阶段未受到实质攻击，仍能保证正常工作，同时入侵检测系统可以检测出部分已知攻击和异常行为。

攻陷阶段。攻击对系统的作用最直接，也最容易被系统发现，该阶段持续时间短且破坏性强。攻击者在该阶段通常按照计划的有效攻击方案对系统实施最大程度的攻击，获取期望的攻击效果。

后攻击阶段。攻击者可能重复利用已知漏洞、已知攻击或预置木马、后门等对系统进行二次破坏或入侵，以持续对系统进行控制和后续破坏。

根据上述分析可以发现，攻击的成功往往具有突发性，即迅速地发生并达到破坏效果[30-31]。相对于系统的整个运行周期，攻陷阶段所占用的时间更加短暂，因而维持高成本的安全性以应对突发的攻击对于系统而言是不可取的。自适应的安全策略能够根据环境威胁水平调节系统的安全性，在可能的攻陷阶段来临之前，提高系统防御能力；短时间地维持高防御能力后，可以通过降低防御能力以降低防御代价。

基于上述分析，本文提出一种自适应的时空多样性联合调度策略——SASTD，该策略的设计思路是依据粗粒度的入侵检测结果和固定调度周期动态改变时间多样性和空间多样性策略的配置，使其跟随威胁水平的变化而改变，在保证系统安全性的前提下，尽可能维持服务质量、降低防御代价。

SASTD 依然需要参考入侵检测的结果评估系统面临的威胁水平，但降低了对入侵检测的准确度

的依赖，尤其在空间多样性加入时，多执行体表决时的不一致结果能够发现入侵检测难以检测出的攻击行为，增强了对未知攻击的防御能力。SASTD 不针对特定的攻击类型，仅依据粗粒度的检测结果对威胁环境进行评估，当威胁水平较高时，提高防御能力，以应对已经发生或即将发生的攻击，形成对攻击广泛的防御能力。

SASTD 包括 2 个主要过程，1) 调度对象候选序列的构建，2) 时空多样性自适应转换流程。过程 1)为过程 2)准备所需要的执行体（集）并对执行体（集）按照轮换顺序进行排序，过程 2)是 SASTD 的主要运行流程，根据系统运行中实时威胁水平的评估，通过调度执行体（集）、改变调度周期等操作动态地调整系统的多样性配置。

3.2 调度对象候选序列的构建

3.2.1 执行体候选序列

基于商用组件（COTS, commercial off-the-shelf）构建执行体可以利用现有的软硬件异构性而避免重新设计实现异构化组件，同时 COTS 提供了较成熟的组件，具有相对完善的功能实现以及较好的兼容性，便于执行体的构建与部署。依据具体的应用场景，给定可以进行异构化的层面（属性）以及每个层面的不同种类（属性值）。例如对于 Web 服务场景而言，可以异构化的属性包括操作系统层、服务器软件层以及应用代码层等，其中操作系统层的属性值又可以包括 CentOS、Ubuntu 以及 Windows 等，同类型操作系统的不同版本在本文中被作为相同的属性值。

对于单个执行体的轮换策略，执行体自身安全性越高，调度前后执行体的异构性越大，对系统的整体安全性提升越有利。本文采取调度异构性优先、执行体安全性其次的决定顺序，安排执行体的轮换次序，以保证在相同的执行体集合下，实现较优的调度顺序。其中调度异构性和执行体安全性的计算采用国家漏洞信息共享平台提供的相关漏洞信息进行度量，通过统计相同漏洞数量计算 2 个执行体的调度异构性，执行体安全性则通过累计所包含漏洞的通用漏洞评分系统（CVSS, common vulnerability scoring system）分值得到。每个执行体的候选执行体构成候选序列，按照先调度异构性 d 最大，再执行体安全性 S 最大的原则经过两轮排序构成。当调度优先级相同时，按照相同概率随机选择轮换上线的执行体。假设执行体 E_i 的候选序列为 $E_1 \to E_2 \to \cdots \to E_n$，其中“$\to$”表示“优先于”，任取 E_i 的候选集中的执行体 E_j, E_k，满足以下条件。

1) 若 $d_{ij} > d_{ik}$，则 $E_j \to E_k$。

2) 若 $d_{ij}=d_{ik}$ 且 $S_j > S_k$，则 $E_j \to E_k$。

3) 若 $d_{ij}=d_{ik}$ 且 $S_j=S_k$，则 E_j 与 E_k 优先级相同。

3.2.2 执行体集优先序列

不同的执行体组合可以构成执行体集 ES。对于 n 个异构执行体，存在 2^n 个执行体集合。文献[32]提出了一种空间多样性系统的执行体集多样性度量方法，实验结果表明，执行体集的多样性与安全性总体上呈正相关的关系。基于该文献中多样性度量方法，按照多样性的大小对所有的执行体集合进行排序，其中当且仅当执行体集合包含至少 2 个执行体时，才参与排序。对于多样性相同的执行体集，按照执行体数量由少到多排序，以 H(ES)表示执行体集 ES 的多样性。综上所述，假设最终执行体集优先序列为 $\mathrm{ES}_1 \to \mathrm{ES}_2 \to \cdots \to \mathrm{ES}_l$，其中“$\to$”表示“优先于”。任取执行体集优先序列中的执行体集 $\mathrm{ES}_i, \mathrm{ES}_j$，满足以下条件。

1) $|\mathrm{ES}_i| > 1$。

2) 若 $H(\mathrm{ES}_i) > H(\mathrm{ES}_j)$，则 $\mathrm{ES}_i \to \mathrm{ES}_j$。

3) 若 $H(\mathrm{ES}_i)=H(\mathrm{ES}_j)$ 且 $|\mathrm{ES}_i| < |\mathrm{ES}_j|$，则 $\mathrm{ES}_i \to \mathrm{ES}_j$。

3.3 SASTD 调度流程

SASTD 调度流程主要处理调度时机的选择和调度对象的选择问题。调度时机选择上，为了尽可能提高防御效率，需加入一定的入侵检测机制，当检测到的异常事件达到一定阈值时，认为攻击者即将攻陷系统。除此之外，系统保持一定的固定调度频率，避免无法被入侵检测发现的攻击对系统造成持久破坏。调度对象选择上，根据一定时间内系统对威胁环境的评估，适当地提高安全性或降低安全性，以使多样性提供的安全性与当前系统的威胁水平相符合。

3.3.1 调度时机选择

SASTD 同时具备调度周期和入侵检测 2 种调度触发机制。当系统连续运行时间达到一个调度周期而未发生调度时，认为该调度周期内系统所受威胁水平在安全防御能力范围之内，可以适当降低安全性需求，从而提高服务质量、降低多样性代价。当系统连续运行时间尚未达到一个调度周期却受到威胁检测阈值时，认为系统当前遭受的威

胁水平超出了防御能力范围，需要提高系统的安全性，即使伴随着一定的服务质量损失和多样性代价提升。

调度周期 C 和威胁检测阈值 D 的初始值设置为经验值 $C_{\max}$ 和 D_0。同时，初始值也标志着该多样性系统的安全性下限，即系统的最低防御能力。

确定调度周期的初始值 $C_{\max}$ 是为了保证系统具备一定的恢复能力。根据经验值可以判断出系统平均实施一次成功攻击所需要的时间，并据此设置 $C_{\max}$。当系统遭受到难以修复的攻击或破坏时，通过一次轮换使系统恢复正常服务。因而 $C_{\max}$ 的确定可根据相同类型的服务系统平均遭受严重攻击的周期确定。另一方面，当系统无法通过继续增加空间多样性而提升安全性时，调度周期 C 将逐步减小，设置 C 的最小值为 $C_{\min}$。$C_{\min}$ 由系统服务稳定性的下限决定，当调度周期继续缩短时，系统几乎无法提供稳定的服务。

依据具体应用情况，可以根据防火墙、入侵检测等工具的检测结果对系统当前威胁水平进行粗粒度的评估。当系统具备空间多样性时，进一步结合冗余执行体表决结果中的不一致现象进行评估。具体地，假设 2 次调度之间传统检测工具发生警报次数为 d_{alert}，冗余执行体表决不一致次数为 d_{vote}，则系统的当前威胁水平为 $\text{Threat} = d_{\text{alert}} + d_{\text{vote}}$。威胁检测阈值 D_0 的确定代表着系统对异常行为的容忍限度，当 $\text{Threat} \geqslant D_0$ 时，认为系统需提高防御能力以避免可能的攻击进一步地深入；当 $\text{Threat} < D_0$ 时，认为当前防御能力足够强，可以尝试降低防御水平以维持服务质量、降低代价。

3.3.2　调度对象选择

调度对象的选择依据触发因素和当前系统防御水平而定，主要解决的问题是如何提升安全性和降低安全性。

时间多样性与空间多样性在提供安全性与多样性代价上存在不同的特点。空间多样性系统在执行体集的异构性较高时，相比时间多样性系统能够提供更高的安全性，然而同时增加了服务响应时延，且执行体集的冗余度（执行体数量）越高，响应时延越大；除此之外，在线执行体数量较多时，也增加了耗能。时间多样性系统的轮换频率越高，安全性越高，但同时对服务稳定性产生一定的副作用；时间多样性几乎不影响服务响应时延，同时在线执行体数量少，耗能较低。因而时间多样性有利于维持服务质量，空间多样性有利于维持高安全性。

在提升安全性的调度中，为了尽可能降低防御成本的增长，采取渐进式的提升方法。图 3 为系统多样性配置随着安全性需求提升的变化顺序。

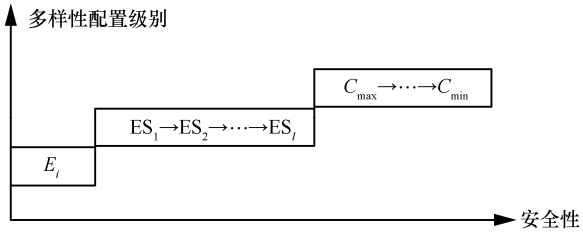


图 3　系统多样性配置随着安全性需求提升的变化顺序

如图 3 所示，系统运行初始上线的执行体数量设置为 1，即在开始运行时，系统处于安全性的最低水平，在没有发生提高安全性的需求时，系统按照纯粹的时间多样性的配置策略，在单一执行体之间进行轮换，轮换对象的上线顺序按照当前执行体的候选序列进行。当发生安全性提高的需求时，系统在纯粹的时间多样性的基础上，叠加空间多样性，采用冗余执行体集代替单一执行体，在安全性需求进一步提高时，按照执行体集优先序列对执行体集进行轮换。若执行体集轮换到多样性最大化的 ES_l 时仍然无法满足安全性需求，则进一步提高多样性配置级别，缩小调度周期，随着安全性需求的继续提升，调度周期不断缩小直到 $C_{\min}$。当安全性需求下降时，按照与上述过程相反的方向改变系统的多样性配置。

3.3.3　SASTD 调度策略

综合调度时机选择和调度对象选择策略的分析，具体调度流程如算法 1 所示。

算法 1　SASTD 调度策略

初始化

构建执行体候选序列;

构建执行体集优先序列;

设置 $C_{\max}$，$C_{\min}$，D_0;

//随机选择执行体 E_i 为系统当前在线执行体 $E\_\text{curr} = E_i$;

//调度周期设置为最大值 $C = C_{\max}$;

运行

while(true)

　//提高安全性

　if (检测警告次数达到检测阈值 D_0)

if (在线执行体数量为 1)
调度 ES_1 上线；
else if (在线执行体集不是 ES_1)
按照安全性序列调度当前执行体集的下一个执行体集上线；
else if (调度周期大于 C_{min})
减小当前调度周期；
if (当前调度周期低于 C_{min})
当前调度周期设置为 C_{min};
end if
else
//安全性已达到最大值
向管理员发出警告；
end if
//降低安全性
else if (连续未调度时长达到当前调度周期)
if (当前调度周期小于 C_{max})
增大当前调度周期；
if (当前调度周期大于 C_{max})
当前调度周期设置为 C_{max};
end if
else if (在线执行体数量大于 1)
if (在线执行体集不是 ES_1)
按照安全性序列调度当前执行体集的前一个执行体集上线；
else
随机选择一个执行体替换当前执行体（集）；
end if
按照安全性序列调度当前执行体的下一个执行体上线；
end if
end if

如算法 1 所示，在初始化过程中，构建每个执行体的候选集从而构成执行体轮换顺序图，并建立执行体集的优先序列。根据经验值以及系统实际安全防御需求设置 C_{max}、C_{min}、D_0 的值，构成系统的防御能力上下限。系统开始运行后，SASTD 实时对系统的受威胁水平进行检测和评估，当系统的检测警告次数达到检测阈值 D_0 时，系统开始按照图 3 所示的顺序向提高安全性的方向转换多样性配置，当安全性达到多样性配置的最大值时，向系统管理员发出警告，同时系统保持当前多样性配置继续运行。当系统连续运行（期间未发生调度）时间达到调度周期时，说明该调度周期内的威胁水平不高于防御能力，系统则按如图 3 所示向安全性降低的方向转换多样性配置。

在整个运行过程中，系统的安全性有可能在最高点和最低点保持运行，但中间值的维持不超过连续 2 个轮换周期。这种设计的原因在于攻击具有突发性，且一旦某一攻击阶段不成功，攻击就无法继续进入下一个攻击阶段，因而可以在维持一个轮换周期后对系统面临的威胁水平进行重新评估并改变安全性需求。

SASTD 所能达到的防御效果如下：在前攻击阶段，以纯粹的时间多样性对单一执行体进行轮换，以较低的防御代价增强系统的不确定性，增大攻击者探测和收集信息的难度；在可能的攻陷阶段出现之前叠加空间多样性，提高系统的防御能力，最大程度降低系统被攻陷的可能性，同时将攻击尽可能拦截至攻陷阶段之前；即使进入后攻击阶段，系统通过周期性轮换能够清除已发生的攻击，降低系统的损失。

4 基于 OpenStack 的 SASTD 系统实现

为了对 SASTD 的有效性进行评估，本文基于 OpenStack 实现了一种 SASTD 管理系统。该系统主要依赖 SASTD 策略对系统上所承载的应用进行动态管理，实现应用对威胁环境的自适应性。基于 OpenStack 的 SASTD 系统结构如图 4 所示。

SASTD 管理系统主要包括中心控制器、应用层输入/输出（I/O, input/output）代理以及相关应用的多样化镜像库。

多样化镜像由服务提供方构建上传至 SASTD 的镜像库，并向中心控制器提供所有执行体的属性、属性值以及其他可能的镜像信息，如安全等级、已知漏洞等，以便计算多样性、执行体安全性等。

中心控制器基于 OpenStack API 调用虚拟机管理功能对 OpenStack 进行二次开发，实现了对服务供应方需求、应用层 I/O 代理以及镜像库的管理和服务。对于服务供应方，中心控制器接收其服务上线以及相关要求的请求，并存储服务供应方上传的镜像和服务信息，代替服务供应方向用户提供服务并进行服务安全性配置的管理。对于应用层 I/O 代理，中心控制器为每个服务创建一个应用层 I/O 代

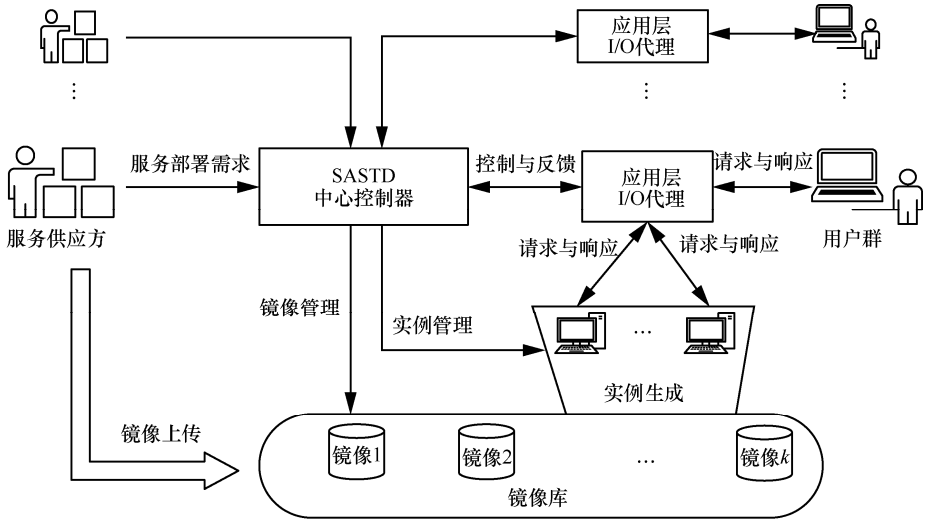


图 4　基于 OpenStack 的 SASTD 系统结构

理，并根据服务信息和安全性要求向应用层 I/O 代理下发安全性配置策略，同时监控应用层 I/O 代理的状态，以便及时替换失联或崩溃的 I/O 代理。中心控制器实时接收应用层 I/O 代理的反馈信息，依据存储的服务需求信息在要求范围按照 SASTD 调度流程（如 3.3 节所述）更新安全性配置，并将更新的配置策略下发；同时，通过调用 OpenStack API 实现虚拟机的创建、删除等操作自动更新 I/O 代理所关联的虚拟机节点。

应用层 I/O 代理作为服务供应方在 SASTD 系统中的代理，是服务的实际提供者，并实现对特定应用服务的直接管理。应用服务启动后，负责接收用户的请求，按照多样性配置决定的分发策略和输出策略，将请求下发至各执行体并收集响应后经过处理返回至用户。除了提供相关服务以外，应用层 I/O 代理接受中心控制器的管理，并向中心控制器反馈当前服务存在的异常情况，例如，裁决不一致次数或入侵检测告警次数达到检测阈值等。中心控制器依据上述反馈信息，对该服务的多样性策略配置进行改变，并将策略变换信息下发至应用层 I/O 代理，然后应用层 I/O 代理按照新的多样性策略继续运行服务。由于应用层 I/O 代理与用户直接连接，因而需要采取必要的安全防护措施，如开启防火墙、设置访问控制、部署入侵检测工具等。即使应用层 I/O 代理失效，中心控制器能够对应用层 I/O 代理进行重建以恢复服务的可用性。

SASTD 系统的使用者包括 2 类，一类为服务供应方，另一类为每个应用面向的用户。SASTD 系统对服务供应方提供的服务实施管理，以实现服务或应用的自适应调节，维持服务或应用的安全性以及高可用性。该系统具有承载多种服务或应用的能力，以基础设施即服务（IaaS, infrastructure as a service）的云服务模式工作。特定服务面向的用户的服务接口由相应的应用层 I/O 代理提供，用户访问相应的 I/O 代理实现对特定应用的访问。

5　实验分析

5.1　实验环境

为便于攻防实验的实施，在 SASTD 系统上部署包含多种不同类型漏洞的 Web 站点 DVWA（damn vulnerable Web application）作为攻击目标。模拟服务提供者提供的多样化的系统属性，包括页面代码、php 相关指令集、sql 相关指令集、服务器软件和虚拟机操作系统 5 种，进行了多样化部署的属性为 sql 指令集操作系统和服务器软件层。具体地，服务器软件层的属性值包括 apache、nginx 以及 IIS（Internet information service），操作系统层包括 Windows、Ubuntu 和 CentOS 等。sql 指令集的异构化主要通过对 sql 语句关键字随机化实现[33]，实验中构造了 2 种不同的指令集。在给定的属性以及属性值范围内，初步筛选出合理的属性值组合，例如 {apache, CentOS, sql_v1}，构成多个异构的能够独立处理服务请求的执行体种类。实验中构造的执行体镜像包括 4 种，具体执行体镜像属性值如表 1 所示。

表 1 执行体镜像属性组成

软件栈层次	执行体 A	执行体 B	执行体 C	执行体 D
页面代码	DVWA v1.10	DVWA v1.10	DVWA v1.10	DVWA v1.10
php 相关指令集	v5.5.12	v5.5.12	v5.5.12	v5.5.12
sql 相关指令集	sql_v1	sql_v1	sql_v2	sql_v2
服务器软件	apache	nginx	apache	IIS
虚拟机操作系统	CentOS	Ubuntu	CentOS	Windows

表 1 中所包含的 5 种属性是 Web 攻击通常涉及的软件栈层面，为了保证实验中攻击具有一定的成功率，仅对部分属性采取了异构化，其他保持了一定的同构性。

漏洞选择从 DVWA 所提供的 8 种漏洞利用中去除了与服务器自身不相关的攻击，例如以服务器为跳板但对服务本身无影响的跨站脚本攻击与跨站请求伪造攻击，共保留了 6 种漏洞以及相关的 10 个具体的攻击行为。攻击包括已知攻击和未知攻击（以是否能够触发入侵检测警报为分类依据），能够触发检测的攻击行为有 6 种，作为已知攻击。具体漏洞以及相应的攻击行为如表 2 所示。

以文件包含漏洞为例，相应场景包括 3 种，分别是利用该漏洞读取后台服务器的敏感文件 etc/shadow，在服务器本地执行命令修改站点访问权限使主页不可访问，执行远程木马文件获取服务器的主机控制权。每种攻击行为针对特定的软件栈构造，对应不同的能够被成功攻击的执行体类型，攻击共模率是指同一攻击能够同时攻陷的执行体在{A,B,C,D}中所占的比例。

作为对比，对纯粹的时间多样性系统（TDS, temporal diversity system）和空间多样性系统（SDS, spatial diversity system）进行了测试。TDS 采用与 SASTD 最低安全等级相同的工作机制，即当检测到的警报到达阈值或到达调度周期时，按照单一执行体轮换顺序（如 3.2.1 节所述）进行轮换。文献[34]也采用了类似的策略，当检测到攻击或达到固定调度周期时均进行一次虚拟机的迁移，实现了基于虚拟机迁移的云中移动目标防御，但本文测试中，TDS 和 SASTD 还引入了调度对象选择策略。SDS 使用了多样性最大的执行体集{A, B, C, D}同时处理相同输入，并收集同一请求的各执行体的响应，当且仅当一致响应数量超过 2 个时才向用户返回一致响应，否则返回错误页面；另外，SDS 运行期间不轮换执行体。根据 3.2.2 节的执行体集合优先顺序，多样性最大的执行体集安全性最高，选取执行体集{A, B, C, D}与 SASTD 进行对比，能够反映出服务质量和代价方面 SASTD 的优势。

5.2 安全性

对系统防御能力的评价上，包含了 Web 扫描和攻击重现 2 种测试方法。其中 Web 扫描过程包含了可能的漏洞的发掘，能够反映系统的漏洞暴露程度，主要评价前攻击阶段中系统的安全性表现。通过攻击再现率评价攻击的可持续性，即在系统运行的不同阶段，攻击能够持续成功重现并对系统产生破坏的可能性。攻击再现率反映出系统在后攻击阶段的防御能力。

5.2.1 Web 扫描

采用开源 Web 扫描工具 xray 分别对 SASTD、

表 2 执行体镜像属性组成

漏洞类型	攻击行为	能够被成功攻击的执行体类型	攻击共模率	攻击类型
暴力破解	猜解密码	A,B,C,D	100%	已知
不安全的验证流程	绕过验证流程	A,B,C,D	100%	已知
sql 注入	泄露数据库其他用户信息	A,B	50%	未知
命令注入	获取服务器操作系统版本	A,C	50%	已知
	发现服务器网段内其他主机	A	25%	已知
文件包含	读取服务器本地敏感文件	C	25%	已知
	在服务器本地执行命令	A,B,C	75%	未知
	执行远程木马文件	A,B,C,D	100%	未知
文件上传	上传木马读取站点目录	A,C	50%	未知
	上传木马在服务器本地执行命令	A,B,C	75%	已知

TDS、SDS 以及静态单一执行体 D 进行扫描，扫描报告所包含的漏洞总数如表 3 所示。SDS 的漏洞数量在 4 种系统中最低，原因在于异构性使多数与操作系统相关的信息无法取得一致响应而不能输出，而该类漏洞在其他 3 种系统均存在一定的可利用时机。

表 3　　Web 扫描结果

系统类型	漏洞数量/个
SASTD	61
TDS	66
SDS	57
执行体 D	66

SASTD 系统在 Web 扫描中暴露出的漏洞少于 TDS 和单一执行体 D。该结果说明 SASTD 系统在前攻击阶段具有一定的隐匿漏洞的效果。由于扫描造成系统威胁水平提升，触发了 SASTD 的安全性提升机制，因而 SASTD 系统对后期扫描中的漏洞利用起到了一定的防御作用，从而减少了扫描报告中的漏洞数量，但在安全性提升之前，存在一定的漏洞利用时机，因而扫描出的漏洞数量多于 SDS。

由于 xray 包含的漏洞扫描类型均存在于站点页面内，而表 1 中 4 种执行体镜像的页面代码不存在异构性，因而 TDS 对攻击面的转移效果未能在表 3 结果中体现。同时，扫描报告所包含的多数漏洞是站点自身的漏洞，所占比例较高，SDS 和 SASTD 的漏洞隐匿作用在表 1 中表现均不突出。

5.2.2　攻击重现

扫描结果中包含的漏洞具有较多同类型漏洞，在本节测试中，采用表 2 中所提供的具有不同共模率的攻击进行测试。由于 SDS 不具备动态性，攻击能够以 100%的可能性再现，因而未加入攻击再现率的实验对比。

对于具有不同共模率的攻击，分别编制自动化攻击脚本在系统运行过程中周期性反复实施攻击，设置系统的入侵检测为较细粒度，即攻击一旦发生系统能够立刻发现，测试得到不同系统对每种攻击的平均攻击重现率，结果如图 5 所示。其中，攻击频率是指调度周期与攻击周期的比值。

图 5 结果显示，对于 2 种系统而言，随着攻击共模率的提升，攻击再现率也呈现升高趋势，说明执行体的异构性有利于减少攻击的再现率。在相同的攻击频率和攻击共模率下，SASTD 攻击重现率总体上低于 TDS，说明 SASTD 的防御能力高于 TDS，能够在后攻击阶段有效地降低系统被再次攻陷的可能。

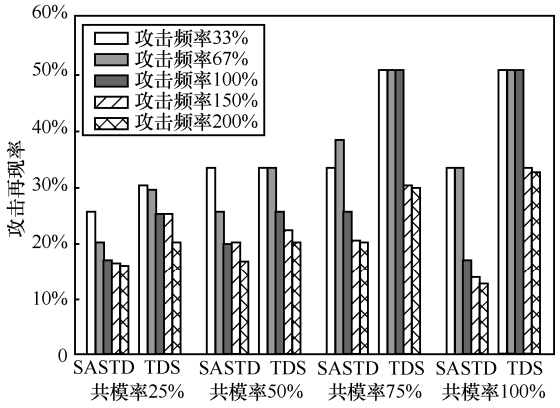


图 5　不同漏洞共模率及攻击频率下 SASTD 和 TDS 的攻击再现率

随着攻击频率的提高，攻击再现率逐渐减小，这是由于攻击频率较高时无效攻击次数较多，部分情形下存在攻击再现率相等的情形，如同一系统在相同共模率、不同攻击频率下，攻击再现率相同，类似情形由于攻击脚本的周期性导致，具有一定的偶然性，不影响上述结论的得出。

5.3　服务质量

服务质量的评估中，综合表 2 中所有的攻击场景编制自动化攻击脚本，轮流实施每个攻击行为，以一次循环为一个攻击周期，测试不同攻击频率下每种多样性系统的响应时延和错误率。为了保证每一个攻击均能够开展，仅在每个攻击周期的最后一次攻击才对系统进行实质的破坏，即关闭 Web 服务。

错误率指标既包含了由于执行体切换引起的服务暂停，也包含了由于攻击导致的服务停止。由于错误率对用户体验影响显著，因而列入服务质量范围内讨论。

如图 6 所示，SDS 相对于 SASTD 和 TDS，由于不具备动态性，一旦被攻击成功，服务就在剩余测试过程中处于瘫痪状态，因而错误率几乎达到 100%，响应时延给出的是服务尚未瘫痪时的平均响应时延，由于表决占用较多时间，因而时延远大于 SASTD 和 TDS。

对于 SASTD 和 TDS，随着攻击频率的提高，2 个系统的响应时延未发生明显变化，而错误率略

有提升，说明攻击频率的提高会降低系统服务的稳定性。SASTD 的响应时延略高于 TDS，而错误率显著低于 TDS，说明 SASTD 会导致服务响应时延增大，但在绝对值上，响应时延的增大并不显著；SASTD 相对于 TDS 显著地抑制了系统的错误率，说明 SASTD 对服务质量的维持能力优于 TDS。

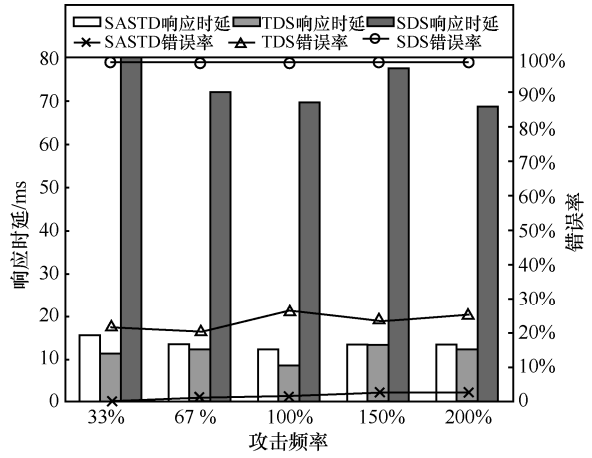


图 6　不同攻击频率下 SASTD、TDS 和 SDS 的响应时延和错误率

5.4　代价

对代价的评估在 5.3 节的实验中获取相关指标的测试结果。由于 SDS 不发生调度，在线执行体数量始终为 4 个，因而未将结果显示在测试结果中。

如图 7 所示，随着攻击频率的提高，SASTD 与 TDS 的调度次数均随之增大，SASTD 的调度次数总体上低于 TDS，结合安全性和服务质量的测试结果，说明 SASTD 能够在较小的调度次数下实现较高的防御能力。

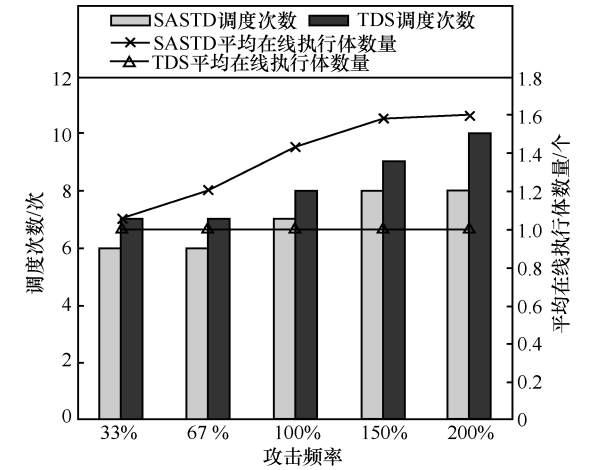


图 7　不同攻击频率下 SASTD 与 TDS 的调度次数和平均在线执行体数量

系统平均在线执行体数量上，SASTD 高于 TDS，且平均执行体数量随着攻击频率的提升呈增大趋势，但测试结果中平均在线执行体数量在攻击频率最高时为 1.4 个，相比于纯粹的空间多样性系统 SDS（在线执行体数量始终为 4 个），在引入空间多样性的条件下，运行代价得到有效控制。

5.5　入侵检测依赖性

为了验证SASTD对入侵检测结果的弱依赖性，本节对 SASTD 的自适应性与入侵检测准确度的关系进行了进一步研究。

在攻击频率为100%的条件下对SASTD和TDS进行测试，统计 2 个系统在不同的入侵检测粒度下全部攻击的再现率以及未知攻击再现率，结果如图 8 所示。

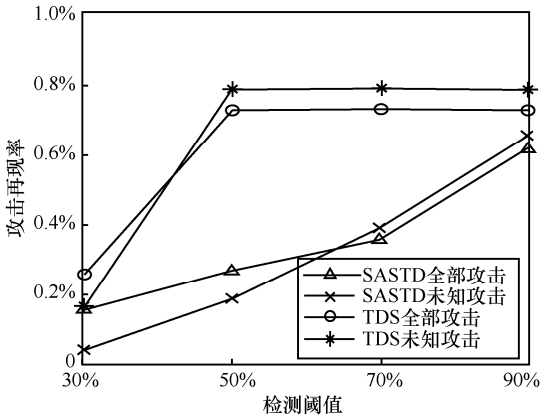


图 8　不同入侵检测粒度下 SASTD 与 TDS 的全部攻击的再现率以及未知攻击再现率

图 8 结果显示，入侵检测粒度越粗，即检测阈值越大，系统对攻击行为的发现越“迟钝”，攻击再现率越高。随着入侵检测阈值的增大，TDS 的全部攻击再现率增长速率为 2.35（检测阈值为 30%~50%），且在检测阈值为 50%时达到最大值后不再变化；而 SASTD 仅为 0.77，且在检测阈值为 90%时攻击再现率仍低于 TDS，说明 SASTD 对入侵检测粒度的依赖性弱于 TDS。SASTD 未知攻击的攻击再现率显著低于 TDS，且变化趋势与 SASTD 全部攻击再现率相似，说明在 SASTD 策略下，未知攻击也能够得到有效的防御。

6　结束语

为了联合时空多样性的优势进行调度策略优化，本文提出一种自适应的时空多样性联合策略。基于攻击的突发性特点，设计了自适应调度策略，

使系统依据环境威胁水平提升或降低防御能力。基于自适应调度策略实现了 SASTD 系统，对安全性、服务质量和代价进行了测试评估。结果表明，相比纯粹的时间多样性系统和空间多样性系统，SASTD 系统能够以较低的防御代价提高了系统防御能力，同时维持了高服务质量。同时，SASTD 相比纯粹的时间多样性系统降低了对入侵检测的依赖。

在当前应用环境下，软硬件存在一定的同源性，易造成攻击在具有相同漏洞的组件间传播扩散。高效地利用软硬件多样性以发挥最大的安全性对传统防御技术有着显著的弥补作用，尤其在应对未知威胁方面。本文所提时空多样性联合调度算法为后续相关工作的开展提供了一定的参考。时空多样性更智能的联合技术[35]有可能成为下一步的研究方向。

参考文献：

[1] 席荣荣, 云晓春, 张永铮, 等. 一种改进的网络安全态势量化评估方法[J]. 计算机学报, 2015, 38(4): 749-758.
XI R R, YUN X C, ZHANG Y Z, et al. An improved quantitative evaluation method for network security[J]. Chinese Journal of Computers, 2015, 38(4): 749-758.

[2] 陈福才, 扈红超, 刘文彦, 等. 网络空间主动防御[M], 北京: 科学出版社, 2018.
CHEN F C, HU H C, LIU W Y, et al. Cyberspace active defense[M]. Beijing: Science Press, 2018.

[3] BANGALORE A K, SOOD A K. Securing Web servers using self cleansing intrusion tolerance (SCIT)[C]//2009 Second International Conference on Dependability. Piscataway: IEEE Press, 2009: 60-65.

[4] OKHRAVI H, COMELLA A, ROBINSON E, et al. Creating a cyber moving target for critical infrastructure applications using platform diversity[J]. International Journal of Critical Infrastructure Protection, 2012, 5(1): 30-39.

[5] DUNLOP M, GROAT S, URBANSKI W, et al. MT6D: a moving target IPv6 defense[C]//2011 Military Communications Conference. Piscataway: IEEE Press, 2011: 1321-1326.

[6] JACKSON T, HOMESCU A, CRANE S, et al. Diversifying the software stack using randomized NOP insertion[M]. Berlin: Springer, 2013.

[7] 仝青, 张铮, 张为华, 等. 拟态防御 Web 服务器设计与实现[J]. 软件学报, 2017, 28(4): 883-897.
TONG Q, ZHANG Z, ZHANG W H, et al. Design and implementation of mimic defense Web server[J]. Journal of Software, 2017, 28(4): 883-897.

[8] WANG C X, DAVIDSON J, HILL J, et al. Protection of software-based survivability mechanisms[C]//2001 International Conference on Dependable Systems and Networks. Piscataway: IEEE Press, 2001: 193-202.

[9] LU K J, SONG C Y, LEE B, et al. ASLR-guard: stopping address space leakage for code reuse attacks[C]//Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2015: 280-291.

[10] GHOURAB E M, SAMIR E, AZAB M, et al. Diversity-based moving-target defense for secure wireless vehicular communications[C]//2018 IEEE Security and Privacy Workshops. Piscataway: IEEE Press, 2018: 287-292.

[11] WANG F Y, JOU F, GONG F M, et al. SITAR: a scalable intrusion-tolerant architecture for distributed services[C]//Foundations of Intrusion Tolerant Systems, 2003 [Organically Assured and Survivable Information Systems]. Piscataway: IEEE Press, 2003: 359-367.

[12] ZHOU L D, SCHNEIDER F B, RENESDSE V R. COCA: a secure distributed online certification authority[J]. ACM Transactions on Computer Systems, 2002, 20(4): 329-368.

[13] POSNETT D, SOUZA R D, DEVANBU P, et al. Dual ecological measures of focus in software development[C]//2013 35th International Conference on Software Engineering. Piscataway: IEEE Press, 2013: 452-461.

[14] SENGUPTA S, CHOWDHARY A, SABUR A, et al. A survey of moving target defenses for network security[J]. IEEE Communications Surveys & Tutorials, 2020, 22(3): 1909-1941.

[15] 蔡桂林, 王宝生, 王天佐, 等. 移动目标防御技术研究进展[J]. 计算机研究与发展, 2016, 53(5): 968-987.
CAI G L, WANG B S, WANG T Z, et al. Research and development of moving target defense technology[J]. Journal of Computer Research and Development, 2016, 53(5): 968-987.

[16] 丁万夫, 郭锐锋, 秦承刚, 等. 容错优先级可提升的抢占阈值容错调度算法[J]. 软件学报, 2011, 22(12): 2894-2904.
DING W F, GUO R F, QIN C G, et al. Preemption threshold scheduling algorithm with higher fault-tolerant priority[J]. Journal of Software, 2011, 22(12): 2894-2904.

[17] NGUYEN Q L, SOOD A. Designing SCIT architecture pattern in a Cloud-based environment[C]//2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops. Piscataway: IEEE Press, 2011: 123-128.

[18] ALAVIZADEH H, JANG-JACCARD J, KIM D S. Evaluation for combination of shuffle and diversity on moving target defense strategy for cloud computing[C]//2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). Piscataway: IEEE Press, 2018: 573-578.

[19] MIR I E, HAQIQ A, KIM D S. A game theoretic approach for cloud computing security assessment using moving target defense mechanisms[C]//Mediterranean Symposium on Smart City Applications. Berlin: Springer, 2017: 242-254.

[20] FENG X T, ZHENG Z Z, MOHAPATRA P, et al. A stackelberg game and markov modeling of moving target defense[C]//International Conference on Decision & Game Theory for Security. Berlin: Springer, 2017: 315-335.

[21] ZANGENEH V, SHAJARI M. A cost-sensitive move selection strategy for moving target defense[J]. Computers & Security, 2018, 75: 72-91.

[22] COLBAUGH R, GLASDS K. Predictability-oriented defense against

adaptive adversaries[C]//2012 IEEE International Conference on Systems, Man, and Cybernetics. Piscataway: IEEE Press, 2012: 2721-2727.

[23] 雷程, 马多贺, 张红旗, 等. 基于网络攻击面自适应转换的移动目标防御技术[J]. 计算机学报, 2018, 41(5): 1109-1131.
LEI C, MA D H, ZHANG H Q, et al. Moving target defense technique based on network attack surface self-adaptive mutation[J]. Chinese Journal of Computers, 2018, 41(5): 1109-1131.

[24] ALAVIZADEH H, KIM D S, JANG-JACCARD J. Model-based evaluation of combinations of Shuffle and Diversity MTD techniques on the cloud[J]. Future Generation Computer Systems, 2020, 111: 507-522.

[25] ALAVIZADEH H, HONG J B, JANG-JACCARD J, et al. Comprehensive security assessment of combined MTD techniques for the cloud[C]//Proceedings of the 5th ACM Workshop on Moving Target Defense. New York: ACM Press, 2018: 11-20.

[26] ALAVIZADEH H, KIM D S, HONG J B, et al. Effective security analysis for combinations of MTD techniques on cloud computing (short paper)[C]//International Conference on Information Security Practice and Experience. Berlin: Springer, 2017: 539-548.

[27] GOUES C L, TUONG A N, CHEN H, et al. Moving target defenses in the helix self-regenerative architecture[M]. Berlin: Springer, 2013.

[28] OKHRAVI H, HOBSON T, BIGELOW D, et al. Finding focus in the blur of moving-target techniques[J]. IEEE Security & Privacy, 2014, 12(2): 16-26.

[29] 刘文彦, 霍树民, 陈扬, 等. 网络攻击链模型分析及研究[J]. 通信学报, 2018, 39(S2): 88-94.
LIU W Y, HUO S M, CHEN Y, et al. Analysis and study of cyber attack chain model[J]. Journal on Communications, 2018, 39(S2): 88-94.

[30] 杨嵘, 张国清, 韦卫, 等. 基于 NetFlow 流量分析的网络攻击行为发现[J]. 计算机工程, 2005, 31(13): 137-139, 164.
YANG R, ZHANG G Q, WEI W, et al. Discovery of network attack behavior based on NetFlow traffic analysis[J]. Computer Engineering, 2005, 31(13): 137-139, 164.

[31] 孙建坡. 基于攻击链的威胁感知系统[J]. 邮电设计技术, 2016(1): 74-77.
SUN J P. The threat perception system based on attack chain[J]. Designing Techniques of Posts and Telecommunications, 2016(1): 74-77.

[32] TONG Q, GUO Y F, HU H C, et al. A diversity metric based study on the correlation between diversity and security[J]. IEICE Transactions on Information and Systems, 2019, 102(10): 1993-2003.

[33] CHEN P, WANG J S, PAN L, et al. Research and implementation of SQL injection prevention method based on ISR[C]//2016 2nd IEEE International Conference on Computer and Communications. Piscataway: IEEE Press, 2016: 1153-1156.

[34] DEBROY S, CALYAM P, NGUYEN M, et al. Frequency-minimal moving target defense using software-defined networking[C]//2016 International Conference on Computing, Networking and Communications. Piscataway: IEEE Press, 2016: 1-6.

[35] 张明悦, 金芝, 赵海燕, 等. 机器学习赋能的软件自适应性综述[J]. 软件学报, 2020, 31(8): 2404-2431.
ZHANG M Y, JIN Z, ZHAO H Y, et al. Survey of machine learning enabled software self-adaptation[J]. Journal of Software, 2020, 31(8): 2404-2431.

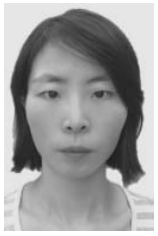
[作者简介]

仝青（1992− ），女，河南郑州人，信息工程大学博士生，主要研究方向为网络空间主动防御等。

郭云飞（1963− ），男，河南郑州人，信息工程大学教授、博士生导师，主要研究方向为网络空间安全等。

霍树民（1985− ），男，山西长治人，博士，信息工程大学副研究员，主要研究方向为网络空间安全、人工智能安全等。

王亚文（1990− ），男，河南郑州人，博士，信息工程大学助理研究员，主要研究方向为云计算安全、网络主动防御等。



茼羽佳（1990− ），女，河北邯郸人，中国人民解放军 32066 部队工程师，主要研究方向为网络安全等。

张凯（1986− ），男，山东肥城人，中国人民解放军 31401 部队助理工程师，主要研究方向为云计算安全等。