文章编号：2096-1618(2024)01-0028-09

基于多变量的执行体动态调度算法

张振嘉， 甘 刚
(成都信息工程大学网络空间安全学院，四川 成都 610225)

摘要：拟态防御技术致力于从内生安全的角度构建安全可靠的系统来解决网络空间中攻防不对称的问题。作为拟态防御中的重要部件，调度模块的关键问题之一是异构冗余体的量化工作。而现有的研究中对异构冗余体的指标不能全面量化，同时调度算法大多将执行体数量设定为某一固定值，对安全性、动态性以及运行效率造成影响。结合调度算法的特性，分析异构冗余体量化过程中的一些重要指标以及这些指标对动态异构冗余构造带来的收益，提出一种多变量负反馈调度算法。通过对多种因素的量化及计算，并在调度过程中对执行体组的状态进行监控，实现执行体数量根据执行体组情况的动态调整。实验结果表明，执行体数量可动态调整提高了动态异构冗余构造的动态性，保持良好的防御能力的同时具有更高的运行效率。

关 键 词：网络空间安全；内生安全；拟态防御；动态异构冗余；量化算法；调度算法；执行体数量
中图分类号：TP393.08 **文献标志码**：A
doi：10.16836/j.cnki.jcuit.2024.01.006

0 引言

随着信息技术的飞速发展和智能设备的不断增加，目前网络空间的安全问题日益严重。数据处理和通信中的弱点、设计错误和后门已成为最大的不确定性[1]。信息系统的安全防御技术分为被动防御和主动防御[2]。现有的防御系统大多是被动防御，需要先了解攻击的来源、攻击的功能、攻击的路径、攻击的行为和攻击机制，作为有效防御的基础[3]。主动防御则着力于提升系统内生的构造安全性、健壮性[4]。邬江兴院士提出的网络空间拟态防御(cyber mimic defense，CMD)技术解决了当前网络空间中存在着广泛的基于未知漏洞和后门等不确定性威胁的内生安全问题。

动态异构冗余(dynamic heterogeneous redundancy，DHR)是 CMD 的核心构造[5]，其构造如图 1 所示。DHR 构造机制引入动态性和随机性，以及多模裁决机制，可以显著提高系统的入侵容忍和容错能力。从攻击者的角度来看，拟态防御系统是一个不确定的系统，提高了系统对非合作攻击的防御能力。

DHR 构造中异构冗余体的调度是 CMD 的关键问题之一，对执行体组进行调度的调度模块是为了接收输出裁决模块的反馈并调整执行体组中的异构冗余体，不断改变系统的运行规律，增加系统内部的不确定性。对异构冗余体进行量化决定了 DHR 构造在调度过程中所选择的异构冗余体组成的执行体组是否为最优执行体组。

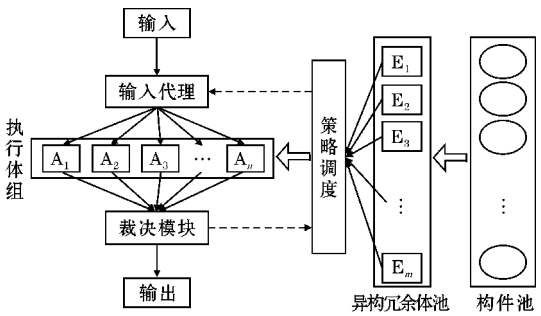

图 1 DHR 构造

本文提出多变量负反馈调度算法(multivariate feedback artificial weighted algorithm，MFAWA)，同时对历史威胁信息、异构度、漏洞威胁度与时间等因素进行量化，将 DHR 构造中执行体组的执行体数量根据当前建立执行体组的状况进行调整。具体贡献如下：从攻击者以及系统效率的角度考虑调度过程中的量化指标，阐述其在调度过程中的用法，避免调度过程中缺乏相关指标的考虑而出现的缺陷；在运行过程中使用合适的执行体数量，实现调度过程中安全性与运行效率综合的良好效果，防止固定数量带来的效率影响以及安全性问题；通过进行碰撞实验和大数裁决系统攻击实验，对 5 种调度方案进行对比评估，结果验证 MFAWA 在保持一定安全性优势的同时在执行效率上相较于以往方案具有更好的表现。

1 相关研究

在 DHR 构造中，调度算法保持系统的高度动态性

收稿日期：2023-03-27
基金项目：四川省哲学社会科学基金资助项目（SC21B034）；四川省科技厅重点研发资助项目(23ZDYF0380、2021ZYD0011)

和不确定性,可以防止攻击者的长期检测和合谋攻击,导致瞬时逃逸。根据调度算法的目标和实现过程,主要对置信度、异构度进行评估。

武兆琪等[6]针对输出裁决进行改进,设计了基于执行体异构度的拟态裁决优化方案,属于正反馈调度。杨林等[7]提出基于历史信息的人工调度算法,该算法在每次防御结算时统计历史威胁出现的次数,调整等待池中异构冗余体的换入概率。李俊等[8]基于拟态防御系统中多模裁决机制的特点,使用漏洞分析方法,提出一种基于执行体安全性的智能仲裁算法。基于执行体的多样性大大提高了系统安全性[9]这一特性,姚文斌等[10]提出最长相异性距离组件选择(maximum dissimilarity,MD)算法和最佳平均相异距离的软件组件选择(optimal mean dissimilarity,OMD)算法。MD 算法在选择软件时始终选取相异距离最长的组件,而 OMD 算法选取具有最佳平均相异距离的软件组件。针对二阶异构度不能整体地表现执行体组的共模漏洞情况,魏帅等[11]提出高阶相似度的相关含义,并设计了一种裁决算法。贾洪勇等[12]基于容斥原理加以分析,给出高阶相似度和高阶异构度的计算方式,引入高阶异构度思想,提出并设计了一种同时考虑历史威胁信息和执行体高阶异构度的调度算法。Wu 等[13]提出一种面向拟态防御系统的基于执行体异构度、性能和历史置信度的随机种子调度算法,综合指标提升了 42.59%。

魏帅等[14]基于安全增益、成本代价以及系统安全性,得出三模冗余容错在安全性和成本代价方面达到最佳的总体效果。Qi 等[15]在拟态网络操作系统(mimic network operating system,MNOS)[16]的研究中提出一种基于反馈的动态感知调度算法,其下一次执行体的数量由系统允许的控制器错误数量决定。李军飞[17]提出一种基于效用的动态灵活调度方法,使用概率密度函数[18]和轮盘法[19],根据当前网络中异常执行器的数量和停机时间确定下一个执行体数量和下一个计划间隔,并根据当前网络环境确定下一步执行体的数量。高明等[20]基于决策反馈的结果,在考虑到动态性和冗余性的情况下,提出一种基于决策反馈的量化算法。该算法以执行体输出的可靠度占比为判决依据,根据前后时刻变化更新调度个数。

在大部分相关文献中,虽然结合移动目标防御等经验提出了相应的量化算法与调度算法,但都缺少其他指标的考虑,并且执行体数量均为固定值。虽然在执行体数量方面有相关研究,但均从执行体历史表现或者当前环境所能容忍的失效率而确定执行体数量,并未考虑到执行体数量对执行体组的影响以及可容忍的最小执行体数量。

2　算法设计

2.1　问题描述

某一含有高威胁度低频出错的异构冗余体与某一低威胁度较高频出错的异构冗余体,在过去研究中将会选择前者。假设因为该漏洞导致逃逸将会出现相对于后者更加重大的损失,即使未发生逃逸,该漏洞的负反馈调节带来的调整也是和低威胁漏洞相同程度的调整。甚至在不针对漏洞进行历史信息更新而仅对异构冗余体更新的情况下,将对未上线的、同样含有该漏洞的其他异构冗余体没有任何调整,结果在之后的调度中对于该高危漏洞依然含有同样的风险。

同时,对于时间方面缺少量化将会导致调度选择更多的异构冗余体加入执行体组,消耗更多的资源,或者严重降低性能。尽管在低负载情况下的代价相对不明显,但一旦处于高负载环境,其中所包含的问题将会随着输入的增多而愈发明显。

对上述问题进行举例,如表 1 ~3 所示。

表 1　漏洞信息表

漏洞编号	威胁度	发生(攻击)频率
1	0.5	0.14
2	1.8	0.11
3	3.7	0.12
4	2.5	0.07
5	5.8	0.08
6	10	0.2
7	4.1	0.13
8	2.9	0.05
9	6.7	0.1

表 2　异构冗余体信息表

异构冗余体编号	所含漏洞	最慢运行时间/ms
1	1,2,3	0.78
2	4,5,6	1
3	7,8,9	1.11
4	1,6,7	1.2
5	2,6,8	1.24
6	3,6,9	1.4

对于不同的执行体组中执行体数量要求,均能得到每个数量对应的最高阶漏洞最低值(若存在多个异构冗余体可选则选择最低频率)并对该种情况进行选取。

表 3 不同执行体数量下的执行体组相关参数

执行体数量	执行体组	最高阶漏洞阶数	最慢运行时间/ms
1	3	1	1.11
2	23	1	1.11
3	123	1	1.11
4	1235	2	1.24
5	12356	3	1.24
6	123456	4	1.4

在执行体数量为 1 或 2 时,发现在同样的最高阶漏洞阶数中,存在 1 号、2 号、3 号共 3 个异构冗余体待选。并在选择的过程中,如果优先考虑异构冗余体所包含漏洞的频率,则异构冗余体 1 始终不是最优解。但从威胁度与最慢运行时间的角度考虑,冗余体 1 号的总漏洞威胁度和最慢运行时间远低于异构冗余体 2 号、3 号。

由表 3 可知,不同执行体的最高阶漏洞阶数与执行体数量的比值从小到大分别为 100%、50%、33%、50%、60% 和 66%。不难看出,在该组候选异构冗余体下,执行体组最佳数量为 3,能满足对于最高阶漏洞的攻击不会发生逃逸。可一旦候选异构冗余体增加或者更换,执行体组最佳数量也将随着异构冗余体组的改变而有所变化。因此,需要相应的调度算法根据异构冗余体及其建立的执行体组来决定执行体组的数量。

假设在执行体数量为 3 的情况下,对 6 号漏洞进行攻击。理想情况下,该执行体组中 1 号、3 号执行体输出正常,2 号执行体输出异常。如果采用执行体历史信息进行更新,而不是漏洞历史信息更新,那么对于 2 号、4 号、5 号、6 号异构冗余体所共同包含的 6 号漏洞不能得到有效地计数。这样,在下一次针对 6 号漏洞进行攻击时,系统将只能判断 2 号异构冗余体的输出效果不理想,却不能发现具有相同漏洞的 4 号、5 号、6 号异构冗余体也不能有效输出。

从上述分析可知,对于候选异构冗余体池的不同情况,其所建立的执行体组的最佳执行体数量也应不同。如使用固定值,将会降低拟态系统的动态性,进而影响安全性与执行效率。

2.2 量化

将从攻击者的角度出发,对漏洞中所考虑的指标以及效率相关指标进行思考与分析,针对 DHR 中的几个主要指标,具体阐述这些指标的用法,对过去研究中存在的问题进行解决,并给出使用以上指标的异构冗余体量化公式。

2.2.1 漏洞历史信息

历史信息对应过去研究中的置信度,记录置信过程中漏洞发生频率。针对过去研究中缺少漏洞发生频率的分析以及如何通过执行体历史信息形成漏洞历史信息的问题,在此根据异构冗余体与漏洞的关系提出一种方法。

根据每个执行体携带有若干种漏洞,每个漏洞可能存在多个执行体中的情况,假设执行体不会出现失效或故障,只有受到针对所含有漏洞的攻击才会失效。那么可以确定,如果一个执行体输出出现问题,必定是由其所含有的某个或某些漏洞导致的。

因此,将异构冗余体所包含的漏洞形成异构冗余体的漏洞信息表,在之后的 DHR 构造每次执行时,对裁决模块判定异常的执行体进行记录,这些执行体所包含的漏洞历史信息均需+1 计数,通过多个异常执行体计数的累加,则被攻击的漏洞所增加的计数将大于其他未被攻击的漏洞。

2.2.2 威胁度

威胁度表示漏洞的严重程度,所有漏洞通过通用漏洞评分系统(common vulnerability scoring system, CVSS)记录其威胁度。CVSS 评分值越高,意味着针对该漏洞攻击的破坏性也将更大。加入这一指标,可以对威胁度更低的漏洞有着更高的容忍度,而针对威胁度更高漏洞的攻击将会更加难以成功。因此,对异构冗余体所含有的所有漏洞威胁度均使用该值表示。

2.2.3 相似度

以往的二阶异构性分析未能考虑高阶异构性,没有分析其对执行体组裁决的影响,且计算方法存在缺陷,难以求出高阶共模性漏洞。攻击者对 DHR 构造进行攻击时,往往并不关注执行体间的异构度情况,而关注执行体组中高阶共生漏洞,只有针对高阶共生漏洞进行攻击,才能取得更高的成功率甚至出现逃逸现象。因此,将不再针对异构度或高阶异构度进行研究分析,而使用高阶相似度替代。

2.2.4 执行效率

由于裁决模块需要收到所有执行体的输出结果才能进行裁决判定,因此在使用同样的调度与裁决算法时,数据分发、裁决与结果输出所消耗时间大致相同,一次运行所消耗的时间基本取决于该执行体组的运行效率。

一个程序每次运行的时间不同,但一定存在最长的运行时间。因此,对于每个异构冗余体的执行效率采用该异构冗余体历史执行时间的最慢值。在之后的运行中对每个执行体的运行进行计时,如果该执行体的本次运行时间超过历史执行时间的最慢值,则对该值进行更新。同时,根据裁决特性,建立的执行体组的运行效率取决于组内最慢执行体的工作时间,一个执行

体组建立的过程中将只记录每个执行体中最大的历史执行时间的最慢值。如果新加入的异构冗余体的历史执行时间的最慢值低于组内该值,则对该执行体组的效率不造成影响。反之,对于执行体组内造成影响,则作为减慢执行体组效率的代价。相比于其他备选的异构冗余体,该执行体加入时需要根据自身数值进行补偿,同时更新执行体组的历史执行时间的最慢值。

2.2.5　计算方式

假设有 m 个备选异构冗余体,这些异构冗余体共包含 n 个漏洞,对于漏洞 b_i,其安全性相关公式:

$$p(b_i)=\frac{h_i\cdot p_i}{\sum_{j=1}^{n}h_j\cdot p_j} \tag{1}$$

该公式表示对于漏洞 b_i,其历史信息经过威胁度加权后,被攻击者选择的概率。

对于某个异构冗余体 E_i,其包含的漏洞集合为 B_i,元素数量为 $k(k\leqslant n)$,则该异构冗余体漏洞损失公式:

$$p(E_i)=\sum_{j=1}^{k}p(b_j)\,(b_j\in B_i) \tag{2}$$

该公式表示对于异构冗余体 E_i,因为其包含的漏洞,导致这个异构冗余体被攻击的概率。归一化:

$$P(E_i)=\frac{p(E_i)}{\sum_{j=1}^{m}p(E_j)} \tag{3}$$

该公式表示仅考虑漏洞的情况下,对于所有备选异构冗余体,该异构冗余体被攻击者选择进行攻击的概率。

将该异构冗余体 E_i 加入后,执行体组高阶相似度归一化:

$$P(S_i)=\frac{S_i}{\sum_{j=1}^{m}S_j} \tag{4}$$

该公式在文献[12]已经提到,表示仅考虑逃逸可能性的情况下,对于所有备选异构冗余体,该异构冗余体被攻击者选择进行攻击的概率。

假设当前执行体组中最长执行时间为 $T_{\max}$,该异构冗余体则时间补偿:

$$t_i=\begin{cases}T_i, & T_{\max}=0\\ 1, & T_i\leqslant T_{\max}\\ T_i/T_{\max}, & T_i>T_{\max}\end{cases} \tag{5}$$

该公式表示当该异构冗余体历史执行时间的最慢值低于组内该值时,加入执行体组将不会产生影响。反之,将根据相应比例计算时间补偿。在执行体组选择第一个异构冗余体加入时,则直接将该异构冗余体的最长执行时间进行计算。

最后,该异构冗余体加入执行体组的损失值:

$$L(E_i)=t_i(P(E_i)+P(S_i)) \tag{6}$$

损失值表示漏洞方面、相似度方面以及执行时间的综合评估。该值越高,表示该异构冗余体的效果越差,被选入执行体组的概率越低。

2.3　调度设计

在调度策略中,根据之前对异构冗余体量化的指标经过计算得到异构冗余体加入执行体组后的损失值,选择损失值最低的异构冗余体加入执行体组。当不再出现高阶相似度漏洞时,根据需要可选择停止加入异构冗余体。此时,异构冗余体加入执行体组时需要满足以下两个条件,直到没有满足条件的异构冗余体为止:最高阶相似度阶数不应增加;执行时间应当比组内最长执行时间小。

符号表示及定义如表 4 所示。

表4　符号表示

符号	定义
$\boldsymbol{\Omega}$	异构冗余体池
N	异构冗余体数量
E_i	异构冗余体 i,$1<i<N$
$\boldsymbol{H}$	漏洞历史信息矩阵
P	漏洞评分矩阵
V	异构冗余体漏洞概率矩阵
$\boldsymbol{S}$	高阶相似度矩阵
T	最慢执行时间
$\boldsymbol{L}$	损失值矩阵

对表 4 中的符号做出以下说明。

定义 1　最慢执行时间 T:对某个异构冗余体 w,其表示其历史表现中的最慢执行时间 T_{w};而对执行体组,则其表示组内各执行体中最慢执行时间的最大值 $T_{\max}$。

定义 2　异构冗余体池:异构冗余体池分为挂起池 $\boldsymbol{\Omega}_{\mathrm{h}}$ 和备选池 $\boldsymbol{\Omega}_{\mathrm{w}}$,分别表示待选的异构冗余体以及不再符合调度标准的异构冗余体。$\boldsymbol{\Omega}=\{E_1,E_2,\cdots,E_N\}$,各异构冗余体之间相互独立。

根据以上定义及计算式,算法如下所示。

算法 1　多变量负反馈调度算法 MFAWA。

输入:调度模式,备选池 $\boldsymbol{\Omega}_{\mathrm{w}}$,挂起池 $\boldsymbol{\Omega}_{\mathrm{h}}$,漏洞历史信息矩阵 $\boldsymbol{H}$,漏洞评分矩阵 P,执行体组 Sel,最小损失值集合 best_l,$T_{\max}$ 初始为 0,$\boldsymbol{\Omega}_{\mathrm{h}}$,Sel 与 best_l 初始为空。

输出:执行体集 Sel

1)计算 L 中最小值所对应执行体,存于矩阵 Lmin

2)for Ei in Lmin

3)　Sel = Ei, Tmax = Tw

4)　if S{Sel} >0

5)　　for Ew in {Ωw- Sel-Ωh}

6)　　if S{Sel} =0 &&(Smax{Sel} <Smax{Sel+Ew}

```
|| Tw >Tmax)
7)          Ωh = Ωh+Ew
8)        else   计算 Lw
9)          if Lw<best_l
10)           best_l←Lw
11)           Sel = Sel+Ew
12)           if Tw>Tmax
13)             Tmax = Tw
14)           end if
15)         end if
16)       end if
17)   end for
18) end if
19) end for
20) 输出 Sel
```

2.4 用例

下面对 MFAWA 运行流程进行简要说明。

步骤 1　选择调度模式,以确定调度将按照 S(Sel) = 0 时停止还是备选池无异构冗余体时停止。初始执行体组 Sel 为空,挑选 $\boldsymbol{L}$ 数值中最小的异构冗余体 E_1 进行调度加入 Sel。若存在多个符合条件的异构冗余体,则将这些不同情况的 Sel 保存,该异构冗余体最慢执行时间赋值给执行体组最慢执行时间。Sel = $\{A_1\}$,进行下一阶段。

步骤 2　调用上一阶段结束时 S(Sel)的值,检查是否符合停止条件。若符合,则进入最后一阶段;若不符合,则继续进行执行体组建立。对将备选池中除执行体组 Sel 中已有执行体和挂起池 Ω_h 中已有异构冗余体外的异构冗余体 E_i 与 Sel 匹配,当 S(Sel) = 0 的条件下,执行体组加入该异构冗余体最高阶漏洞阶数提升或该异构冗余体最慢执行时间大于执行体组最慢执行时间,则将该异构冗余体加入挂起池。计算 $L = T(S(Sel+\{E_i\})+V_i)$,选择 L 最小的异构冗余体加入 Sel,该异构冗余体最慢执行时间大于执行体组最慢执行时间,则更新执行体组最慢执行时间;若 L 最小值不唯一,将对应的异构冗余体分别加入 Sel。Sel = Sel + $\{A_i\}$,将 Sel 送入算法进行下一阶段,重新回到步骤 2。

步骤 3　将 Sel 输出,若存在多个 Sel,则选取执行体组最慢执行时间最小的一组进行输出。

2.5 算法分析

目前,针对执行体组策略的评估方法有很多,如基于博弈模型对攻防双方进行收益量化[21]、攻击成功累计时长[22]、成功控守目标个数[23]、共模漏洞个数[12]等指标作为评判标准。本文将以最高阶漏洞阶数与执行体数量的比例和漏洞评分作为评判标准。

在最高阶漏洞阶数与执行体数量的比例方面,当高阶漏洞数量为 0,裁决方式为大数判决,攻击者一次只能对一个漏洞进行攻击的情况下,在该执行体组中,执行体不会出现失效或者故障,将不可能通过对已知漏洞的攻击实现逃逸。同时,高阶漏洞数量为 0,执行体组将不再加入会使最高阶漏洞阶数增加的异构冗余体。意味着在公式最高阶漏洞阶数与执行体数量的比例中,分子将会固定在某值,而分母随着异构冗余体的增加而不断增大,这一数值将会不断降低。

在调度过程中,直接影响到漏洞评分的指标是执行体的最慢运行时间与漏洞历史信息。对于异构冗余体最慢运行时间,若低于执行体组最慢运行时间,将不会对执行体组的运行效率造成影响,在时间的计算上也与其他同样低于执行体组最慢运行时间的异构冗余体相同。在计算的过程中,若在漏洞历史信息与高阶相似度也相同的情况下,所含漏洞威胁度更低的异构冗余体的损失值将会低于所含漏洞威胁度更高的异构冗余体的损失值。在调度过程中,将会选择所含漏洞威胁度更低的异构冗余体。

相比于异构冗余体最慢运行时间低于执行体组最慢运行时间的异构冗余体,在漏洞历史信息与高阶相似度也相同的情况下,高于执行体组最慢运行时间的异构冗余体如果要被调度,其所含漏洞威胁度应当更低,才能够平衡执行效率更差带来的时间上的补偿。同样,在时间与高阶相似度相同的情况下,漏洞的威胁度与历史信息呈负相关,这也意味着包含高威胁度漏洞的异构冗余体相比包含高威胁度漏洞的异构冗余体将会更难被选进执行体组中。同时漏洞威胁度不同,在发生有关这些漏洞的攻击之后,其负反馈的效果也是不同的。若两个漏洞的评分之比为 1∶5 时,当两者均发生一次分别针对这两个漏洞的攻击后,其漏洞的安全性评分分别增加 1、5,在这之后连续四次对低评分漏洞进行攻击时,两者漏洞的评分变化才达到一致。可以看出,对于严重程度更高的漏洞的容忍程度更低,在调度的选取过程中,将会更低频率、更少次数地选择含有高危漏洞的异构冗余体。

同时,执行体数量根据执行体组的状态变化选择系统最高阶漏洞阶数与执行体数量之比更低、执行效率更高的异构冗余体进行调度,增加了执行体组组合方式的数量,使算法的动态性得到了提升。

3 仿真实验

一个优秀的调度算法往往需要根据拟态构造的裁

决机制进行相应的调整。目前,学者已提出了很多裁决算法,如基于二进制文件的裁决算法[24]、基于异常值的裁决算法[25]、大数裁决算法[26]、一致表决算法[27]、基于高阶异构度的裁决算法[11]等。由于其通用性,目前 DHR 构造使用大数裁决算法多于其它算法。此方法的基本思想,就是以执行体输出相同数量中最多的结果作为判定结果。因此,本文对大数判定算法进行改进,并建立相应的验证模型,对算法准确度以及执行效率进行验证。

3.1　实验设计及改进

为验证 MFAWA 的有效性,将与基于高阶异构度的执行体动态调度、基于历史记录信息的动态调度、先进先出、随机调度 4 种调度算法进行对比,实验分别为碰撞实验和大数裁决系统攻击实验。

碰撞实验由文献[7]提出,其核心思想是通过对攻击原子和防御原子的碰撞率来衡量系统的安全性。碰撞率越高,系统越安全。实验步骤如下:

Step1　等待池和运行池初始化。

Step2　构造威胁输入。

Step3　进行动态调度,从等待池中选取异构冗余体填入运行池。

Step4　进行碰撞检测和计数。

Step5　返回 Step2,重复进行多次实验。

Step6　计算平均碰撞概率。

Step7　改变威胁数目,返回 Step1。

对于大数裁决系统攻击实验,在文献[12]的历史信息矩阵的构造中,直接将该次攻击所针对的漏洞用于漏洞历史信息表的更新。对于防守方而言,并不能够准确地找到是何种漏洞造成的执行体的失效。因此,为更加符合实际情况,对于针对漏洞的历史信息更新,本文均采用前文提到的更新方式。调整后的实验算法如算法 2 所示。

算法 2　大数裁决系统攻击实验算法。

输入:异构冗余体池 Ω,损失值矩阵 L,漏洞历史信息矩阵 H,调度轮次 M,漏洞评分矩阵 P,漏洞选取概率矩阵 A

输出:系统平均攻破率

1)攻破次数 c0=0

2)for j in {1,2,…,M}

3)　　根据威胁选取策略构造威胁矩阵 A

4)　　执行算法 1,获得执行体组 Sel

5)　　cj=0

6)　　for Ei in Sel

7)　　if　Ei 所含漏洞和 T 中攻击原子发生碰撞

8)　　cj=cj+1

9)　　End if

10)end for

11)if cj ≥ Nsel÷2

12)　　c0=c0+1

13)end if

14)将此轮发生碰撞的执行体依次将所含漏洞中的次数+1,对 H 进行更新

15)end for

16)输出系统攻破率 c0÷M

现实环境下,大部分的攻击很难服从均匀分布。原因在于,从攻击者的角度出发,针对不同漏洞的攻击方法的数量、使用的难易度以及成功进行攻击之后所取得攻击效果也不同。同时,对于有历史信息这一指标的量化算法,在均匀分布中的防御效果与先进先出、随机调度的效果差距不大,非均匀分布也更加符合现实情况。所以,本次研究中,随机装载中威胁概率分布将不仅服从均匀分布,对漏洞攻击的概率也将与威胁度呈正相关。

几个基本假设如下:

(1)攻击者每次随机使用一种攻击方式进行攻击。若该攻击方式能同时攻破半数以上处于运行池中的执行体,则拟态系统被攻破。

(2)假设防御方拥有应对所有攻击采用漏洞的防御策略,除异构冗余体所包含的漏洞外,其余攻击原子均能够进行有效防御。

(3)假设整个攻防过程中不会有新的异构冗余体产生,异构冗余体在攻防开始之前已被置入异构冗余体池中且其所包含的漏洞不再改变。

(4)异构冗余体将不会出现故障失效等其他非针对漏洞攻击导致的输出问题。如果异构冗余体在进入执行体组工作的过程中出现输出问题,必定是由于针对该执行体所包含漏洞的攻击导致。

仿真中相关实验参数设定如下:

(1)对于需要确定执行体数量的算法,因现有研究大多数为 5 模异构冗余体集,故执行体调度为 5 模异构冗余体集,即每次从异构冗余体池中调度 5 个异构冗余体到执行体组中。异构冗余体池容量为 20,即每次调度将有 20 个异构冗余体可供选择。在碰撞实验中异构执行体容量参考文献[7]中参数进行设置;在大数裁决系统攻击实验中,当威胁数大于 100 时异构执行体容量设定为威胁数/10,且最小不低于 10。

(2)异构冗余体的防御/漏洞原子、攻击过程中漏洞均为随机生成,分别采用均匀分布和对漏洞攻击的概率将与威胁度呈正相关的正态分布。

(3)实验采用蒙特卡罗方法进行测试,每种实验环境下测试 10000 次,最后记录实验结果。

实验程序用 MATLAB 编写。为更好地量化实验结果,碰撞实验中用碰撞率表示攻击时执行体组内各执行体状况,该值越高,执行体调度越安全;大数裁决系统攻击实验中用系统攻破率表示系统被攻破的概率,该值越低,系统越难被攻破;用系统攻破时所使用的漏洞评分的平均值表示平均漏洞得分,平均漏洞得分越低,代表攻破时对系统造成的破坏程度越低;仿真时间表示调度过程中执行体组的最慢执行时间,该值越高,表示执行体组效率越低。

3.2 安全性实验

3.2.1 碰撞率

在碰撞实验中,分别测试了随机生成和威胁度生成 2 种环境,得到的实验结果分别如图 2 和图 3 所示。

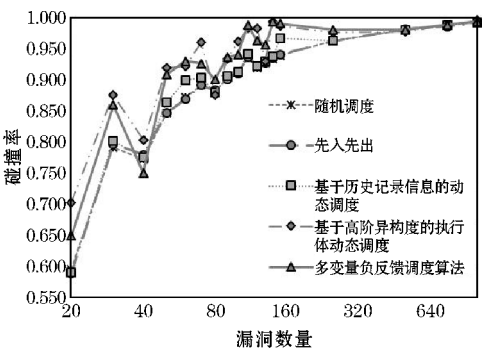

图 2 随机生成下碰撞率测试

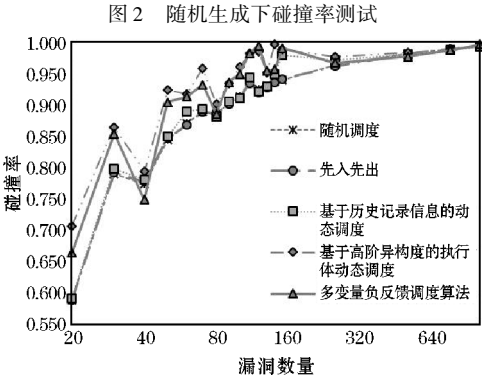

图 3 威胁度生成下碰撞率测试

3.2.2 系统攻破率

大数裁决系统攻击实验同样测试了描述的 2 种环境,得到的实验结果分别如图 4 和图 5 所示。

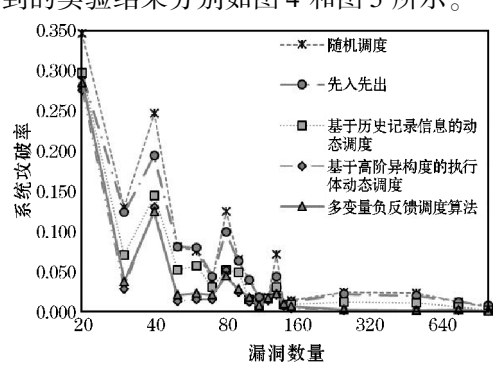

图 4 随机生成下系统攻破率测试

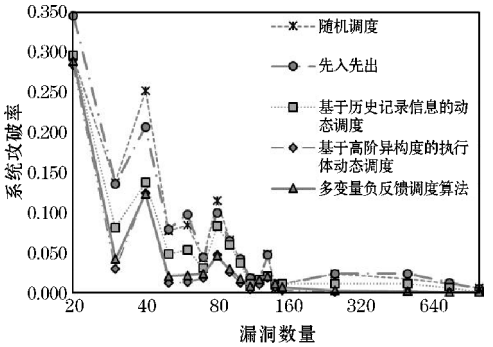

图 5 威胁度生成下系统攻破率测试

3.2.3 平均漏洞得分

根据大数裁决系统攻击实验中系统被攻破的情况,对攻破时所使用漏洞的评分进行统计,结果分别如图 6 和图 7 所示。

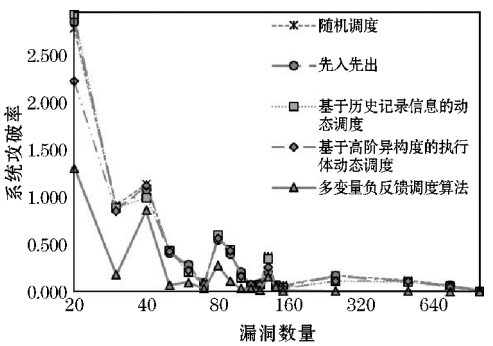

图 6 随机生成攻破时漏洞评分统计

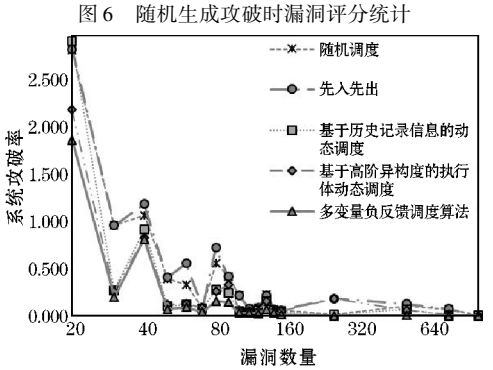

图 7 威胁度生成攻破时漏洞评分统计

3.3 性能测试

根据以上实验参数配置以及大数裁决系统攻击实验中的情况,对所选执行体组的最慢执行时间进行统计,结果分别如图 8 和图 9 所示。

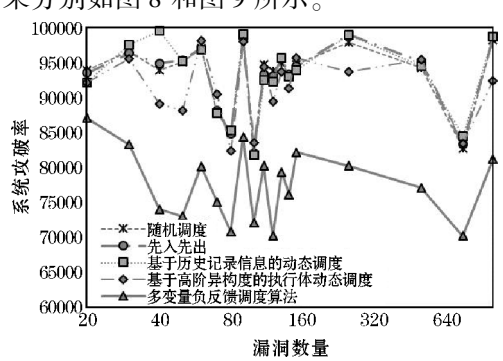

图 8 随机生成下仿真时间统计

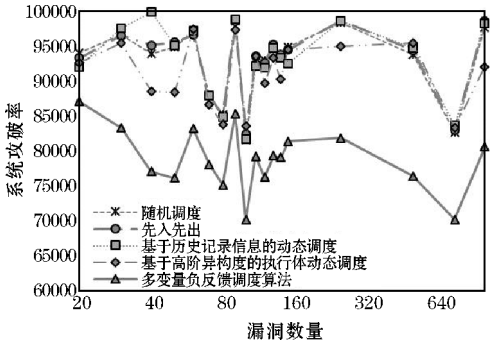

图 9　威胁度生成下仿真时间统计

根据以上实验结果,可以得到以下结论:

(1)在碰撞实验中,多变量负反馈调度算法和基于高阶异构度的执行体动态调度算法的碰撞率基本上高于其他算法。

(2)在大数裁决系统攻击实验中,由于考虑到高阶相似度这一因素,多变量负反馈调度算法的系统攻破率仅高于基于高阶异构度的执行体动态调度算法,但始终低于其他算法。这是因为本文的多变量负反馈调度算法还考虑到时间因素,导致系统攻破率略高于高阶异构度的执行体动态调度算法。

(3)因为考虑了漏洞威胁度这一指标,在系统攻破的情况下,多变量负反馈调度算法的漏洞威胁度低于其他算法,在系统被攻破时不会产生较其他算法更严重的漏洞被攻击者利用。

(4)同时在随机调度下,并不会使加入历史信息考虑的算法优于其他算法。其原因在于,对于漏洞选取服从均匀分布的情况,更新的历史信息表不能使调度算法选择更优的调度方式,其选择方式与先入先出以及随机调度几乎相同;而在漏洞选取服从基于漏洞威胁度的分布等非均匀分布时,经过一定次数后,历史信息中漏洞的计数将基本与漏洞选取的概率趋于一致。因此,在这种条件下,基于历史信息的调度算法将会体现出优势。

(5)由算法性能测试结果可明显看出,因为考虑到时间因素,多变量负反馈调度算法在执行体组建立时将会对异构冗余体的时间进行评估,从而选择执行效率高的异构冗余体加入。在大数裁决系统攻击实验中,执行时间的考虑也使本文的算法中执行体组的执行时间较其他算法更低,为系统带来更高的运行效率。

4　结束语

研究拟态防御系统中的调度算法,分析在异构冗余体量化过程中的一些重要指标,针对现有的执行体调度算法主要将执行体数量设定为某一固定值的问题,提出一种多变量负反馈调度算法。算法通过对多种因素的量化计算,并且在调度时对执行体组的状态进行监控,实现了执行体数量根据执行体组进行动态调整。仿真实验结果表明,MFAWA 在保持良好的防御能力的同时具有更快的运行速度。后续将基于本文所做工作,对相应的裁决算法根据高阶异构度的特点结合其他指标进行研究更新,以达到拟态防御中全过程的优化。

参考文献:

[1] Rios Insua David, Couce-Vieira Aitor, Rubio Jose A, et al. An Adversarial Risk Analysis Framework for Cyberse-curity[J]. Risk analysis: an official publication of the Society for Risk Analysis, 2019, 41(1):16-36.

[2] Jajodia S, Ghosh A K, Swarup V, et al. Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats[J]. Springer Ebooks, 2011, 54.

[3] 邬江兴. 网络空间拟态防御研究[J]. 信息安全学报, 2016(4):10.

[4] 戴人杰, 柴小丽, 邵培南, 等. 拟态通用运行环境外部表决机制研究[J]. 计算机系统应用, 2021, 30(10):180-186.

[5] 扈红超, 陈福才, 王禛鹏. 拟态防御 DHR 模型若干问题探讨和性能评估[J]. 信息安全学报, 2016, 1(4):40-51.

[6] 武兆琪, 张帆, 郭威, 等. 一种基于执行体异构度的拟态裁决优化方法[J]. 计算机工程, 2020, 46(5):12-18.

[7] 杨林, 王永杰, 张俊. FAWA:一种异构执行体的负反馈动态调度算法[J]. 计算机科学, 2021, 48(8):284-290.

[8] 李俊, 王志浩, 陈迎春. 一种基于执行体安全性的智能仲裁算法[J]. 通信技术, 2021, 54(4):929-937.

[9] 韩进, 臧斌宇. 软件相异性对于系统安全的有效性分析[J]. 计算机应用与软件, 2010, 27(9):273-275.

[10] 姚文斌, 杨孝宗. 相异性软件组件选择算法设计[J]. 哈尔滨工业大学学报, 2003(3):261-264.

[11] 魏帅, 张辉华, 苏野, 等. 面向拟态防御系统的高阶异构度大数判决算法[J]. 计算机工程, 2021, 47(5):30-35.

[12] 贾洪勇, 潘云飞, 刘文贺, 等. 基于高阶异构度

的执行体动态调度算法[J]. 通信学报,2022,43(3):233-245.
[13] Wu Z, Wei J. Heterogeneous Executors Scheduling Algori-thm for Mimic Defense Systems[C]. 2019 IEEE 2nd International Conference on Computer and Communi-cation Engineering Technology(CCET). IEEE,2019.
[14] 魏帅,于洪,顾泽宇,等. 面向工控领域的拟态安全处理机架构[J]. 信息安全学报,2017,2(1):54-73.
[15] Chao Qi,Jiangxing Wu, Hongchao Hu,et al. Dynamic-scheduling mechanism of controllers based on security policy in software-defined network [J]. Electronics Letters,2016,52(23).
[16] Hongchao Hu,Zhenpeng Wang,Guozhen Cheng, et al. MNOS: a mimic network operating system for software defined networks[J]. IET Information Security,2017,11(6).
[17] 李军飞. 软件定义网络中拟态防御的关键技术研究[D]. 郑州:战略支援部队信息工程大学,2019.
[18] Parzen Emanuel. On Estimation of a Probability Density Function and Mode[J]. The Annals of Mathematical Statistics,1962,33(3).
[19] Adam Lipowski,Dorota Lipowska. Roulette-wheel selection via stochastic acceptance[J]. Physica A: Statistical Mechanics and its Applications, 2011,391(6).
[20] 高明,罗锦,周慧颖,等. 一种基于拟态防御的差异化反馈调度判决算法[J]. 电信科学,2020,36(5):73-82.
[21] 丁绍虎,齐宁,郭义伟. 基于 M-FlipIt 博弈模型的拟态防御策略评估[J]. 通信学报,2020,41(7):186-194.
[22] 蔡雨彤,常晓林,石禹,等. 动态平台技术防御攻击的瞬态效能量化分析[J]. 信息安全学报,2019,4(4):59-67.
[23] 张兴明,顾泽宇,魏帅,等. 拟态防御马尔可夫博弈模型及防御策略选择[J]. 通信学报,2018,39(10):143-154.
[24] 吴正江,姚琪,冯四风,等. 基于数据库二进制日志的竞赛式仲裁优化方案[J]. 计算机工程,2021,47(5):24-29.
[25] 高振斌,贾广瑞,张文建,等. 基于异常值的拟态裁决优化方法[J]. 计算机应用研究,2021,38(7):2066-2071.
[26] Choi J,Goh K I. Dynamics of consensus formation on multiplex networks: The majority-vote model [C]. APS March Meeting 2018. American Physical Society,2018.
[27] Bass J. Voting in Real-time Distributed Computer Control Systems [J]. University of Sheffield, 1995,20(1):93-102.

A Multivariate Dynamic Scheduling Algorithm for Heterogeneous Executor

ZHANG Zhenjia, GAN Gang
(College of Cybersecurity,Chengdu University of Information Technology,Chengdu 610225,China)

Abstract: Mimic defense technology aims to create a secure and dependable system that addresses the issue of asymmetric attack and defense in cyberspace. As an important part of the mimic defense,one of the key problems of the scheduling module is the quantification of the heterogeneous executor. The current research on the indicators of the heterogeneous executor cannot fully achieve quantification,and most scheduling algorithms fix the number of execution entities, which impacts security,dynamic ability,and operational efficiency. Combined with the characteristics of the scheduling algorithm,this paper analyzes some significant indicators in the process of heterogeneous executor quantification and the benefits of these indicators on the construction of dynamic heterogeneous executor and proposes a multivariate dynamic scheduling algorithm for the heterogeneous executor. By quantifying and calculating various factors and monitoring the status of the executive body group during the scheduling process,the number of executive bodies can be dynamically adjusted according to the situation of the executor group,and verified by simulation experiments. The experiment demonstrates that the number of executors can be dynamically adjusted to improve the dynamic nature of dynamic heterogeneous redundancy construction, maintain good defense capability and have higher operational efficiency.

Keywords: cyberspace security; endogenous safety and security; mimic defense; dynamic heterogeneous redundancy; quantitative algorithm; scheduling algorithm; number of executing entities