文章编号:1671-6833(2024)02-0072-08

云原生环境下基于移动目标防御的 ReDoS 防御方法

扈红超[1], 张帅普[2], 程国振[3], 何威振[3]

(1.郑州大学 中原网络安全研究院,河南 郑州 450001;2.郑州大学 网络安全学院,河南 郑州 450001 3.信息工程大学 信息技术研究所,河南 郑州 450001)

摘 要:针对云原生环境中正则表达式拒绝服务(ReDoS)攻击的防御方式存在效率低、无法进行主动防御的问题,提出了基于移动目标防御(MTD)技术的 ReDoS 攻击防御方法。首先基于云原生环境下的微服务应用特点,对攻防双方的行为进行了分析;其次,基于 Kuberneters 设计了基于 MTD 的防御系统,并提出基于拓扑信息和请求到达速率的动态和静态的多维微服务权重指标、基于排队论的服务效率判断指标以及轮换时机选择方法来指导关键微服务的选择和关键微服务的轮换时机;最后,给出了基于异构度和服务效率的多维指标 MTD 异构轮换算法,并使用 Python 进行了仿真,结果表明:所提算法防御时延比动态伸缩缩短了 50%左右;并且防御开销在第一次攻击之后趋于平稳,不会持续增长。

关键词:微服务; ReDoS; 移动目标防御; 异构; 正则表达式

中图分类号: TP301.6;TP302.1;TP302.7 **文献标志码**:A **doi**:10.13705/j.issn.1671-6833.2023.05.009

随着云原生技术的发展,微服务架构在云环境中的应用也更加广泛[1]。微服务架构[2]将应用程序划分成一组松耦合的细粒度服务,相互独立的同时又相互协作构成微服务链。由于共享微服务的存在使得微服务链之间存在交叉[3],单个服务的性能可以影响整个系统,导致攻击被放大,进而导致更大的经济损失[4]。

正则表达式拒绝服务(regular expression denial of service, ReDoS)攻击者可以设计请求流量,从而利用正则表达式的超线性漏洞发起拒绝服务攻击[5]。该攻击的三要素为可回溯的正则表达式引擎、具有超线性复杂度的正则表达式和非法输入[6]。在云原生环境中 ReDoS 攻击可以通过共享微服务等特殊微服务获取更大的攻击收益[4]。

目前针对云原生环境中的 ReDoS 攻击的防御方式主要分为 2 种:一种利用云原生弹性扩容对受影响的服务进行攻击缓解;另一种在出现 ReDoS 攻击之后对服务进行漏洞检测和修复。前者可以保证服务的正常运行但无法阻止攻击对资源的消耗;后者能够阻止攻击者对同一漏洞的反复利用,但是会造成服务的中断,同时该种方法采用事后加固的方式进行防御,缺乏主动防御能力。

针对弹性扩容方法的研究主要通过动态伸缩微服务资源来缓解攻击的影响。如 Yu 等[7]首先讨论了在云原生环境中 DoS 攻击防御的可能性;其次指出云平台强大的资源调度能力使服务可以在攻击持续期间维持正常运行,从而达到防御目的。Li 等[4]基于入侵检测机制将实例资源隔离,分别处理白名单请求和其他请求,通过生成不同数量的容器最大可能地保证二者的服务质量。Yuan 等[8]考虑了云环境资源收费的特点,使用线性规划给出了攻击期间保证服务质量的最优资源分配。

针对漏洞检测和修复的研究分为基于静态分析检测技术和基于模糊测试的检测技术[9]。前者的优点是检测速度快,缺点是存在误报以及无法分析包含扩展漏洞的正则表达式;后者的优点是可以较好地支持扩展正则表达式的检测,缺点是无法检测隐藏较深的 ReDoS 漏洞(即攻击字符串的长前缀生成问题),并且检测效率低。由于上述 2 种方法在检测效率和实用性上存在不足,多数防御者选择在

收稿日期:2023-03-03;**修订日期**:2023-05-10

基金项目:国家自然科学基金资助项目(2072467);国家重点研发计划项目(2021YFB1006200, 2021YFB1006201)

作者简介:扈红超(1982—),男,河南商丘人,郑州大学教授,博士,主要从事网络空间安全、网络主动防御方面的研究,E-mail:sharpfe@163.com。

引用本文:扈红超,张帅普,程国振,等. 云原生环境下基于移动目标防御的 ReDoS 防御方法[J]. 郑州大学学报(工学版),2024,45(2):72-79. (HU H C, ZHANG S P, CHENG G Z, et al. ReDoS defense method based on moving target defense in cloud-native environment[J]. Journal of Zhengzhou University (Engineering Science), 2024, 45(2): 72-79.)

攻击结束后对危险正则表达式进行修复[5],然而由于正则表达式的广泛应用,多数正则表达式都存在一个或者一个以上的漏洞[10],单个系统也往往存在多个正则表达式。Cloudflare 公司的报道[11]指出:当其公司出现 ReDoS 攻击漏洞后对 Web 应用防护系统组件中的 3 868 条正则表达式规则进行了人工检查。并且以上提到的防御方法只能在关闭当前服务的情况下进行,因此会造成服务中断。

综上所述,弹性扩容的防御方法可以保证微服务运行时的质量,但是无法阻止攻击者对同一漏洞的反复利用。针对正则表达式漏洞的检测和修复技术能够在一定程度上阻止攻击,但是存在假阳性、假阴性和效率低等问题。

针对以上问题,本文提出采用移动目标防御(moving target defense, MTD)方法来防御微服务中的 ReDoS 攻击,MTD 作为一种主动防御技术[12],能够动态轮换云原生环境下的异构微服务应用,能够有效增强云原生应用的主动防御能力,因此可以有效防御云原生环境下的 ReDoS 攻击。首先,本文在对攻防行为进行分析的基础上,在 Kubernetes 云平台上设计了基于 MTD 的防御系统框架;其次,基于排队论模型对微服务应用进行建模,提出了基于服务效率的最佳的轮换时机选择方法,为了全方位判断微服务的运行状态,本文基于图论模型对微服务进行评估,提出静态和动态 2 种指标来指导关键微服务的选择;最后,本文使用 Python 对提出的微服务轮换算法进行仿真实验。实验结果表明:本文提出的防御方法在保证服务性能的同时成功防御了 ReDoS 攻击,也降低了防御开销。

1　攻防分析

本节首先分析了 ReDoS 的攻击流程,然后给出了本文提出的防御模型。

1.1　云原生环境中的 ReDoS 攻击

云原生环境中 ReDoS 攻击者除了使服务瘫痪之外,扩大攻击的影响也十分重要[13]。ReDoS 攻击者的目标是尽可能多地找到存在超线性漏洞正则表达式的微服务,然后选择对整个系统影响最大的微服务发起攻击。图 1 给出了攻击的流程。

漏洞扫描。攻击者在入侵存在正则表达式超线性漏洞的微服务后对其中的正则表达式进行分析,设计出可以引起正则表达式漏洞的字符串,如图 1 中的漏洞探测路线。

微服务选择。攻击者筛选出存在漏洞的微服务集合后,通过微服务之间的信息传递收集微服务间的关联信息,对这几个微服务的影响进行评估,以达到最大的攻击收益。如图 1 中的 B。

漏洞利用。向服务发送带有能触发漏洞的字符串请求,进行 ReDoS 攻击。

假设攻击者总是有办法得到有关微服务漏洞的信息,这在主动防御的攻击假设中是常见的。

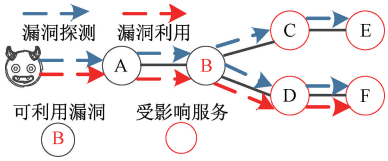

图 1　攻击流程

Figure 1　Attack flow

1.2　基于 MTD 的防御模型

MTD 作为一种主动防御技术,通过变化的攻击面迷惑对手,其主要特点为多样化、动态性和冗余性[14]。本文基于以上 MTD 技术的特点,设计了如图 2 所示的 MTD 防御模型。异构资源池是关键微服务的异构备份,在检测到服务质量异常时,从异构资源池选出与正在运行的微服务实例异构度最大的备份将微服务替换,从而使得 ReDoS 攻击者失去目标漏洞导致攻击失败。

服务质量检测模块。在初始化阶段负责选出关键微服务,运行时该模块负责保证关键微服务的服务质量,当检测到服务效率异常时通知异构备份选择模块进行流量重定向。

异构备份选择模块。该模块主要负责在异构资源池中选出与现有微服务存在最大异构度的模块进行路由重定向和最优备份选择。

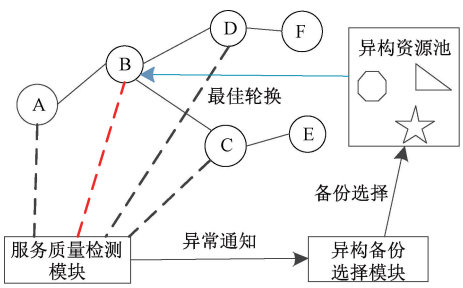

图 2　防御模型设计

Figure 2　Defense model design

2　系统架构设计

为了说明本文提出防御模型的可行性,基于 Kubernetes 容器云[15]平台给出了模型的具体实现方案。如图 3 所示。

防御系统工作流程如下。首先统计集群中微服务的相关信息,根据收集的信息生成每个微服务的紧密性、中心性和必要性等度量指标,从而找出关键

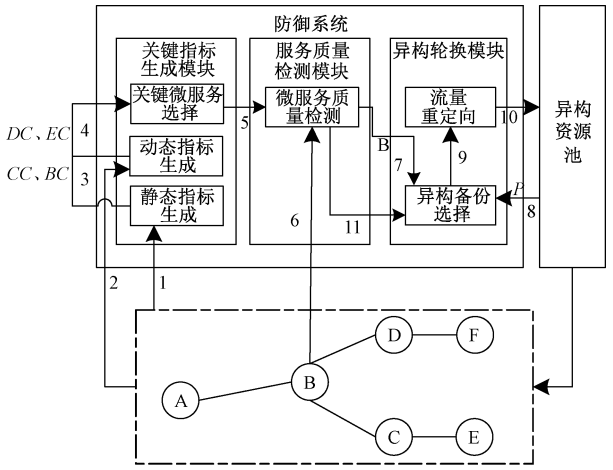

图 3 基于 Kubernetes 的系统设计实现方案

Figure 3 System design and implementation scheme based on Kubernetes

微服务并对其服务效率进行监控,当出现异常时选出与当前实例异构度最大的备份进行流量重定向,最后根据备份的服务效率选择最优备份进行轮换。

静态指标生成。系统初始化阶段负责从集群拉取当前所有微服务的拓扑快照,获取单个微服务与拓扑相关的静态属性,包括紧密性和中心性。

动态指标生成。运行时阶段定期从集群拉取当前所有微服务的请求到达速率和服务速率,生成微服务的动态属性包括中心度和权重。

关键微服务选择。综合静态指标和动态指标找出关键微服务。

异构备份选择。当关键微服务的服务效率出现异常时选出异构度最大的备份微服务,并根据各备份的服务效率选择最优备份进行替换。

2.1 基于图论的关键微服务选择方法

为了平衡防御开销和防御效果,找出关键微服务,本节基于图论给出单个微服务的权重量化指标。

中心性 BC。该指标表示一个节点出现在其他节点最短路径的次数[16]。由于微服务环境中不存在最短路径,所以将服务 m 的 BC 值定义为 $BC(m)=\sum_{i=1}^{i=n}\frac{S_{i,m}}{S_i}$,其中,$S_i$ 表示微服务链总数;$S_{i,m}\in\{0,1\}$,取 1 时表示该微服务链穿过微服务 m,反之取 0。

紧密中心性 CC。紧密中心性表示节点到其他节点的最短跳数[17]。对于单个微服务链来说,上游微服务受到攻击会导致下游全部微服务受影响,所以服务 m 的 CC 值可以表示为 $CC(m)=\sum_{v,m\in S}\frac{1}{d_d(v,m)}$,其中,$d_d(v,m)$ 表示 m 以及和 m 处在同一微服务链下游的微服务 v 的距离,当 m 下游没有微服务时,$d_d(v,m)=0.5$。

中心度 DC。节点的重要程度,代表击溃整个系统的下限,是研究网络鲁棒性的重要指标[18]。在微服务环境中服务的请求到达速率越快,则代表该节点具有较高的中心度,因此该节点的故障会严重影响请求服务质量。服务 m 的 DC 值定义为 $DC(m)=\sum_{v\in V}\frac{\lambda_m}{\lambda_v}$,其中,$V$ 代表所有微服务;λ_v 代表服务 v 的请求到达速率。

根据上述指标就可以求出微服务 m 所占权重 ED,记作 $ED(m)=\frac{\beta BC(m)\cdot\alpha DC(m)}{\gamma CC(m)}$,该指标和 m 对应的下游微服务数量、直接相连的微服务数量以及请求到达速率成正比。其中,β、α、γ 分别为 3 个指标的加权值,可由管理人员自行调整。

得到微服务的经济权重就可以将关键微服务选择表达为一个整数线性规划。

$$\begin{cases}\max\sum_{m\in V}ED(m)x_m;\\ \text{s.t.}\ \sum_{m\in V}x_m\leqslant k;\\ x_m\in[0,1],m\in V。\end{cases}\tag{1}$$

目标函数是为了找到 k 个具有最大权重的服务。约束条件 1 确保在现有服务之中选择服务。在约束条件 2 中,x_m 取 1 时代表其是共享服务,反之取 0。由此得到了由 k 个具有最大 ED 值的共享微服务组成的集合,这些服务将作为部署本文提出的异构调度算法的节点。

2.2 异构微服务应用选择指标

为了提高备份与实例之间不存在相同漏洞[19]的概率,本节基于异构度给出了每个备份被选择的概率。提取异构备份的 P 元特征,如操作系统、设计语言、设计团队以及函数库依赖等,表示为 $\{d_1,d_2,d_3,d_4\}$[20]。将 2 个 P 元特征对应位置差的平方和定义为 2 个异构备份的距离。将其作为判断二者之间异构度的标准,距离越大则代表 2 个异构备份差异越大。

当轮换发生时每个异构备份被选择的概率 P_m 可由式(2)求出。

$$P_m=\frac{D(P_i,P_j)}{\sum_{j=1}^{n}D(P_i,P_j)}=\frac{\sum_{s=1}^{s=n}(P_i(s)-P_j(s))^2}{\sum_{j=1}^{n}D(P_i,P_j)}。\tag{2}$$

式中:s 表示特征索引;n 表示特征数量。对于微服务 P_i,备份 P_j 与 P_i 的异构度越大,该备份 P_j 被选择的概率越高。

2.3　基于排队论的轮换时机选择方法

为了及时发现服务效率的异常,本节基于排队论 M/M/c 模型对请求在微服务系统的服务过程进行建模,并给出微服务服务效率的计算公式。建模时用到的相关变量如表 1 所示。

表 1　建模和算法设计时用到的变量

Table 1　Variables used in modeling and algorithm design

数学符号	含义
B	服务繁忙率
μ	单个容器的请求服务速率
B_n	服务 n 的繁忙率
B_x	备份异构服务的繁忙率
ρ	请求到达速率和服务速率之比
λ	请求到达速率
ω	微服务的时间权重
T	请求的平均等待时间
η	等待队列长度权重
$cost$	单个实例化容器花费

研究人员通过观察得出结论,请求的服务时间具有相同的幂率分布并且与相互到达的时间无关,请求的到达服速率从泊松分布[21]。基于上述结论,本文使用 M/M/c 排队论模型对该系统进行建模是合理的。这意味着微服务请求到达速率服从多个泊松分布的和,请求的服务速率满足指数分布,请求队列无限长。

在容器云环境中微服务的请求到达速率取决于该微服务上游的容器的数量 λ_n,如式(3)所示。

$$\lambda_n = n\lambda。 \tag{3}$$

此外每个容器的服务速率为 μ,对于有 c 个容器并且处在 n 个微服务链交叉点的微服务来说,服务速率如式(4)所示。

$$\mu_n = \begin{cases} n\mu, n \leqslant c; \\ c\mu, n > c。 \end{cases} \tag{4}$$

使用 ρ 表示到达和服务速率的比值:

$$\rho = \frac{n\lambda}{\mu_n}。 \tag{5}$$

当 $\rho \leqslant 1$ 时,该排队论系统的时系统处在稳态,此时系统中有 t 个请求的概率可由式(6)计算。

$$Q_n = \begin{cases} \dfrac{(c\rho)^t}{n!}Q_0, t \leqslant c; \\ \dfrac{c^c\rho^n}{t!}Q_0, t > c。 \end{cases} \tag{6}$$

式中:Q_0 代表队列中没有请求的概率,可由式(7)求出。

$$Q_0(\rho, c) = \left[\sum_{n=0}^{c-1}\frac{(c\rho)^n}{n!} + \frac{(c\rho)^c}{c!\ (1-\rho)}\right]^{-1}。 \tag{7}$$

当服务速率为 μ 时,平均队列长 L_q 可由式(8)求出。

$$L_q = \sum_{t=c+1}^{\infty}(n-c)Q_n = \sum_{k=1}^{\infty}kQ_{k+c} = \frac{(c\rho)^c\rho}{c!\ (1+\rho)^2}Q_0。 \tag{8}$$

此外,根据 Little 公式,可以求出服务速率为 μ 时微服务的平均等待时间:

$$T(\mu) = \frac{1}{\lambda}\left(c\rho + \rho\frac{(c\rho)^c}{c!}\frac{Q_0}{(1-\rho)^2}\right)。 \tag{9}$$

由于不同微服务对于性能的需求不同,本文将服务繁忙率 B 定义为

$$B = \omega T + \eta L。 \tag{10}$$

2.4　基于多维指标的异构轮换算法

当检测到服务质量下降时,启动异构轮换算法。

算法 1　异构轮换算法。

输入:某共享微服务的请求到达和服务繁忙率;

输出:最优缓解策略。

① While (True)
② If ($B_n > B_u$)
③ type = StartRotate ();
④ If (type = = 0) then
⑤ num = GetServingContainer (B_n);
⑥ CreateContainer(num);
⑦ Else
⑧ num = GetServingContainer (B_n);
⑨ CreateContainer (num, type);
⑩ Endif;
⑪ While ($B_n < B_u$)
⑫ ReduceContainerbyOne ();
⑬ endwhile。

算法负责监视所有共享微服务的繁忙率 B_n,当 B_n 超过阈值 B_u 时,启动异构轮换算法,将请求平均分配到异构备份中,通过检测得到最优异构备份进行复制,将服务效率异常的实体进行修复。

算法 2　最优备份选择算法。

输入:异构备份的请求到达和服务速率;

输出:异构备份的种类。

① for R requests
② redirect R to HeterogeneousService;
③ B_x = GetHeterogeneousBusy ();
④ Endfor;
⑤ For B_n in B_x
⑥ If ($B_n < 0.6B_{n+1}$)
⑦ Return n;
⑧ Else return 0。

当轮换算法 startRoate( )启动时,会把所有的请求平均重定向到异构微服务备份中,并且计算所有备份异构微服务备份的繁忙率 B_x,如果有异构微服务备份的服务繁忙率小于其他微服务的繁忙率,则认为该备份不存在当前攻击的漏洞,返回备份类型;如果所有备份的服务繁忙率无差别,则认为当前攻击是利用交叉点微服务的流量放大作用进行的 DDoS 攻击,则随机选择该微服务的备份进行复制,达到资源扩容的目的,使服务正常运行。本文认为当存在一个备份的服务繁忙率低于其他备份的 60%时,2 个备份的服务繁忙率区别较大。

3 对比实验与分析

3.1 实验参数设置

根据 Li 等[4]的研究设置的 ReDoS 攻击持续期间请求到达速率期望和攻击未发生时相同,仿真实验拓扑由 3 条微服务链构成,因为本文关注的是关键微服务的属性,所以对微服务的数量没做过多要求,每条微服务链包含 6 个微服务,仿真实验拓扑图如图 4 所示,其中关键微服务处在 3 条微服务链的交叉点,各链上除共享微服务外请求服务速率分别为 $\mu_1=3$ 请求/s,$\mu_2=2$ 请求/s,$\mu_3=2$ 请求/s,请求到达速率分别为 $\lambda_1=2$ 请求/s,$\lambda_2=1$ 请求/s,$\lambda_3=1$ 请求/s。共享微服务 a 的请求服务速率为 $\mu_a=8$ 请求/s,共享微服务 b 的请求服务速率为 $\mu_b=5$ 请求/s。由此可求出共享微服务的队长期望和服务时间期望,实验开始阶段将每个微服务的容器数量设置为 1。从而求得 B_u。单个微服务可以服务的最大请求服务速率为 10 请求/s,恶意请求服务时间是正常请求的 8 倍。异构资源池中异构备份的数量为 3 个,每个异构备份的 P 元特征组分别为 $P_1=(0,1,0)$,$P_2=(1,0,0)$,$P_3=(1,0,1)$,仿真实验持续期间每 50 min 模拟一次持续 5 min 的 ReDoS 攻击,使用 CPU 利用率作为防御效果的可视化指标。

3.2 必要性指标的有效性实验

如图 4 所示,共有 2 个共享微服务:共享微服务 a 和微服务 2。为了验证本文提出的经济权重的有效性,根据实验参数设置计算 2 个微服务的权重,其中 β、α、γ 的值设置为 1。求得 $BC(a)=1$,$CC(a)=1/6$,$DC(a)=4/25$,故共享微服务 a 的权重为 $ED(a)=24/25$;同理求得 $BC(2)=1/3$,$CC(2)=1/5$,$DC(2)=1/25$,微服务 2 的权重为 $ED(2)=1/15$。

本次分别针对微服务 a 和微服务 2 进行了仿真攻击,由于串联服务台排队论系统后者的请求到达速率和上一个系统的服务时间有关[17-18],本文选择

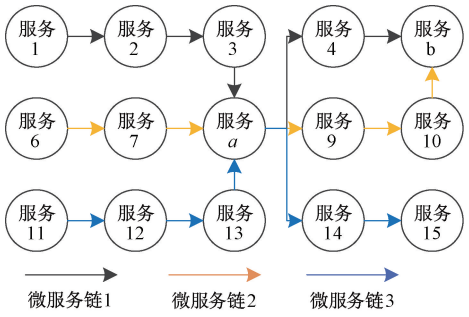

图 4 仿真实验拓扑图

Figure 4 Topology diagram of simulation experiment

下游服务 CPU 利用率变化作经济损失的可视化指标。为了表明影响的一般性,将微服务 2、微服务 4 和微服务 10 在攻击前后的变化绘制成图 5~7 曲线图。

对比图 5 和图 6 可知,ED 值高的微服务在受到攻击时其下游微服务 CPU 利用率下降较快,这是请求阻塞导致的。由于与服务商按照实例的运行时间计费资源没有充分利用导致用户产生了严重的经济损失。对比图 6 和图 7 可知,距离受攻击微服务越远的服务受攻击的影响越小。

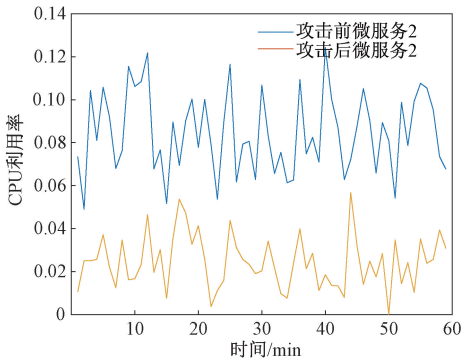

图 5 微服务 2 在攻击前后的 CPU 利用率

Figure 5 CPU utilization of Microservice 2 before and after the attack

3.3 防御开销

本节将从防御开销和防御安全性 2 个方面将所提出的异构轮换算法与文献[3]提出的缓解算法进行对比。非异构扩容防御的经济损失 E 计算公式如下:

$$E = N \cdot cost \cdot T'。 \tag{11}$$

式中:N 为维持服务质量所需的容器数量,根据服务效率不断变化;T'表示备份存活的时间;$cost$ 表示单个容器存活单位时间的花费,设为 1。

由图 8 可知,在不考虑异构度只是根据服务质量对 ReDoS 攻击进行缓解时,随着攻击次数的增加,防御开销呈线性增长,如图 8 所示,这是由于周期发起的 ReDoS 攻击具有相似的攻击强度和攻击时间。

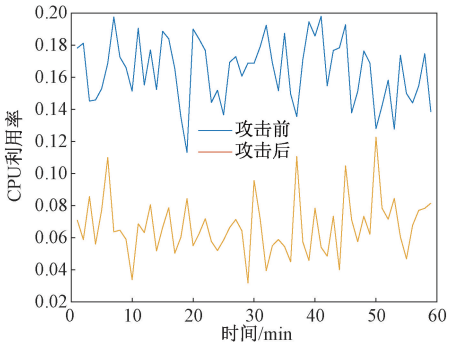

图 6　微服务 4 在攻击前后的 CPU 利用率

Figure 6　CPU utilization of microservice 4 before and after the attack

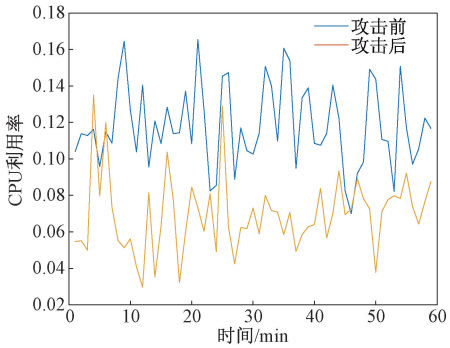

图 7　微服务 10 在攻击前后的 CPU 利用率

Figure 7　CPU utilization of microservice 10 before and after the attack

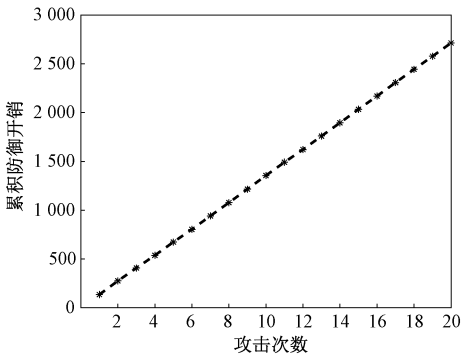

图 8　累积防御开销随攻击次数的变化

Figure 8　Cumulative defense overhead varies with the number of attacks

由于开发异构备份的花费比普通备份多,所以异构扩容带来的经济损失 E 计算公式如式(12)所示。其中 α 为异构备份相较于普通备份开销的系数。

$$E = N \cdot \alpha cost \cdot T'。\tag{12}$$

当 $\alpha=6$ 时,异构扩容随着攻击次数的经济损失累加如图 9 所示,相比于非异构扩容虽然单个异构备份的花费更高,但是由于可以快速地轮换掉存在 ReDoS 漏洞的微服务使攻击的影响时间变短,并且在之后攻击中没有带来额外的开销,所以带来的经济损失远小于非异构轮换。

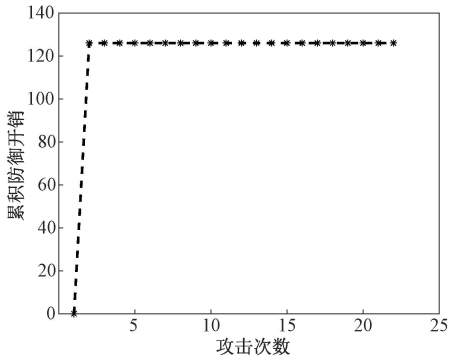

图 9　异构扩容的累积防御开销随攻击次数的变化

Figure 9　Cumulative defense overhead of heterogeneous scaling varies with the number of attacks

3.4　异构调度算法的防御效果

本文提出的防御方法核心思想是当检测到微服务出现异常时启动流量重定向,并根据异构备份的表现选取合适的异构备份进行防御。由图 10 可知,当攻击开始时,3 号异构服务的 CPU 利用率比其他 2 个备份服务低,这是由于异构服务 3 不存在此次 ReDoS 攻击所针对的正则表达式超线性漏洞,所以 3 号备份较其他备份的 CPU 利用率低,故此对该异构备份进行复制。

对比图 10 和图 11 中这 2 种防御方式下 CPU 的利用率变化情况可知,异构调度不仅能对攻击产

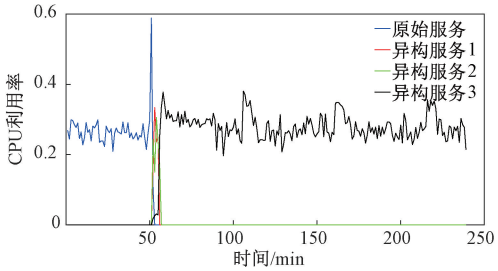

图 10　启动异构度轮换算法各备份的 CPU 利用率

Figure 10　CPU usage of each backup when the heterogeneity rotation algorithm is enabled

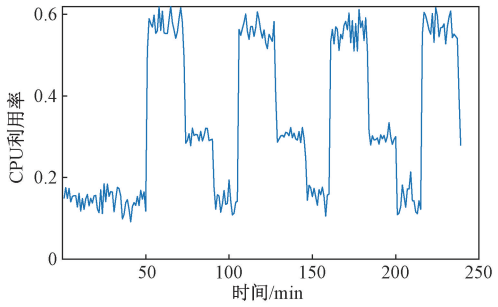

图 11　不考虑异构轮换算法各备份 CPU 利用率

Figure 11　CPU utilization of each back up without considering heterogeneous rotation algorithms

生免疫作用提高云平台的安全性(只出现一次CPU利用率的突变),还可以极大地缓解攻击带来的影响(CPU利用率突变时间短)。

4 结论

本文首先基于图论和微服务架构的特点提出了计算单个微服务权重的公式,并基于MTD技术提出了ReDoS的缓解防御方法。首先,该方法破坏了ReDoS攻击的攻击载体,使其不能持续进行欺诈资源消费。其次,通过仿真实验验证了该防御方法的有效性。该方法的重点是异构微服务备份的选择,未来工作的重点将集中在如何更加细致地分析瓶颈微服务是否出现异常,以及优化异构服务备份的选取策略,同时加深对入侵检测知识的学习。

参考文献:

[1] 岳猛, 王怀远, 吴志军, 等. 云计算中DDoS攻防技术研究综述[J]. 计算机学报, 2020, 43(12): 2315-2336.
YUE M, WANG H Y, WU Z J, et al. A survey of DDoS attack and defense technologies in cloud computing[J]. Chinese Journal of Computers, 2020, 43(12): 2315-2336.

[2] 辛园园, 钮俊, 谢志军, 等. 微服务体系结构实现框架综述[J]. 计算机工程与应用, 2018, 54(19): 10-17.
XIN Y Y, NIU J, XIE Z J, et al. Survey of implementation framework for microservices architecture[J]. Computer Engineering and Applications, 2018, 54(19): 10-17.

[3] 张宇鹏, 吴自力, 陈鸣, 等. 面向交叉微服务链的任务调度优化[J]. 西安电子科技大学学报, 2021, 48(6): 32-39.
ZHANG Y P, WU Z L, CHEN M, et al. Optimization of task scheduling oriented to cross microservice chains[J]. Journal of Xidian University, 2021, 48(6): 32-39.

[4] LI Z, JIN H, ZOU D Q, et al. Exploring new opportunities to defeat low-rate DDoS attack in container-based cloud environment[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(3): 695-706.

[5] LI Y, SUN Y, XU Z, et al. RegexScalpel: regular expression denial of service (ReDoS) defense by localize-and-fix[C]//31st USENIX Security Symposium (USENIX Security 22). Atlanta: USENIX Association, 2022: 4183-4200.

[6] KIRRAGE J, RATHNAYAKE A, THIELECKE H. Static analysis for regular expression denial-of-service attacks[C]//International Conference on Network and System Security. Cham: Springer, 2013: 135-148.

[7] YU S, TIAN Y H, GUO S, et al. Can we beat DDoS attacks in clouds?[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(9): 2245-2254.

[8] YUAN B, ZHAO H, LIN C, et al. Minimizing financial cost of DDoS attack defense in clouds with fine-grained resource management[J]. IEEE Transactions on Network Science and Engineering, 2020, 7(4): 2541-2554.

[9] 沈宇桔. 正则表达式复杂度攻击自动化检测技术研究[D]. 南京: 南京大学, 2019.
SHEN Y J. Research on automatic detection technology of regular expression complexity attack[D]. Nanjing: Nanjing University, 2019.

[10] LI Y T, CHEN Z X, CAO J L, et al. ReDoSHunter: a combined static and dynamic approach for regular expression DoS detection[C]//USENIX Security Symposium. Atlanta: USENIX Association, 2021: 3847-3864.

[11] The Cloudflare Blog. Details of the cloudflare outage[EB/OL]. (2019-07-02)[2022-12-11]. https://blog.cloudflare.com/details-of-the-cloudflare-outage-on-july-2-2019/.

[12] 张晓玉, 李振邦. 移动目标防御技术综述[J]. 通信技术, 2013, 46(6): 111-113.
ZHANG X Y, LI Z B. Overview on moving target defense technology[J]. Communications Technology, 2013, 46(6): 111-113.

[13] OLIVO O, DILLIG I, LIN C. Detecting and exploiting second order denial-of-service vulnerabilities in web applications[C]//Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2015: 616-628.

[14] LEI C, ZHANG H Q, TAN J L, et al. Moving target defense techniques: a survey[J]. Security and Communication Networks, 2018, 2018: 1-25.

[15] PAHL C, BROGI A, SOLDANI J, et al. Cloud container technologies: a state-of-the-art review[J]. IEEE Transactions on Cloud Computing, 2019, 7(3): 677-692.

[16] FREEMAN L C. A set of measures of centrality based on betweenness[J]. Sociometry, 1977, 40(1): 35.

[17] KANG M S, GLIGOR V D. Routing bottlenecks in the internet: causes, exploits, and countermeasures[C]//Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2014: 321-333.

[18] MAGONI D. Tearing down the internet[J]. IEEE Journal on Selected Areas in Communications, 2003, 21(6): 949-960.

[19] 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.
JIA H Y, PAN Y F, LIU W H, et al. Executive dynamic scheduling algorithm based on high-order heterogeneity [J]. Journal on Communications, 2022, 43(3): 233-245.
[20] 曾威, 扈红超, 李凌书, 等. 容器云中基于 Stackelberg 博弈的动态异构调度方法[J]. 网络与信息安全学报, 2021, 7(3): 95-104.
ZENG W, HU H C, LI L S, et al. Dynamic heterogeneous scheduling method based on Stackelberg game model in container cloud[J]. Chinese Journal of Network and Information Security, 2021, 7(3): 95-104.
[21] 刘海芳. 两服务台串联排队系统[D]. 长沙: 中南大学, 2007.
LIU H F. Two service stations in series queuing system [D]. Changsha: Central South University, 2007.

ReDoS Defense Method Based on Moving Target Defense in Cloud-native Environment

HU Hongchao[1], ZHANG Shuaipu[2], CHENG Guozhen[3], HE Weizhen[3]

(1. Zhongyuan Network Security Research Institute, Zhengzhou University, Zhengzhou 450001, China; 2. School of Cyber Science and Engineering, Zhengzhou University, Zhengzhou 450001, China; 3. Information Technology Research Institute, University of Information Engineering, Zhengzhou 450001, China)

Abstract: In addressing the inefficiencies and limitations in proactive defense against Regular Expression Denial of Service (ReDoS) attacks in cloud-native environments, we have developed a defense method based on Moving Target Defense (MTD) technology. Initially, we analyzed the behaviors of both attackers and defenders within microservice applications characteristic of cloud-native environments. Subsequently, leveraging Kubernetes, we designed an MTD-based defense system. This system incorporates dynamic and static multi-dimensional microservice weight indices based on topology information and request arrival rates, as well as service efficiency judgment indices based on queue theory. It also includes a method for selecting the timing of key microservice rotations to guide the selection and rotation timings of critical microservices. Finally, we introduced a multi-dimensional MTD heterogeneous rotation algorithm, grounded in heterogeneity and service efficiency, and conducted simulations using Python. Experimental results indicate that our proposed algorithm reduces defense latency by approximately 50% compared to dynamic scaling and that defense costs stabilize after the initial defense against an attack, preventing continuous growth.

Keywords: microservices; ReDoS; moving target defense; heterogeneous; regular expression