SaaS 云环境下基于容器指纹匿名的网络欺骗方法

李凌书[1*], 邬江兴[1], 刘文彦[1]

[1]国家数字程控交换中心 解放军信息工程大学 郑州 中国 450001

摘要 随着云计算技术在各重点行业领域的普及推广和企业级 SaaS 业务规模不断扩大, 云环境的安全问题也日益突出。针对目标云资源的定位是网络攻击的前置步骤, 网络欺骗技术能够有效扰乱攻击者网络嗅探获得的信息, 隐藏重要网络资产的指纹信息(服务端口、操作系统类型等)。然而由于虚假指纹和真实设备之间往往存在对应关系, 高级攻击者可以通过多维信息定位到伪装过的目标设备。基于容器指纹匿名的欺骗方法的原理是试图将重要的敏感业务隐藏到大量普通的非敏感业务中, 利用多种欺骗技术的组合来对抗网络侦察。批量处理的匿名算法会损失云服务快速便捷的特性, 云服务需要采用一种实时强的连续匿名方法。为应对重要云资源被定位追踪的问题, 本文提出一种基于容器指纹匿名的网络欺骗方法, 通过修改云资源池中容器的指纹满足匿名化标准, 制造虚假的云资源视图, 提高攻击者网络侦查与嗅探的难度。为降低容器指纹修改的开销和可能导致服务延时, 提出一种基于语义等级的范畴属性度量方法, 并作为容器指纹匿名算法的优化目标。鉴于需要修改伪装的容器指纹信息是一个持续产生的数据流, 为实现容器指纹的实时在线快速匿名, 提出一种基于数据流匿名的动态指纹欺骗算法 CFDAA, 通过时延控制和簇分割实现容器指纹的快速修改, 保证在线容器始终满足 k-匿名和 l-多样性。实验结果表明, 所提方法能够在开销可控的情况下, 有效提高攻击者定位目标云资源的难度。

关键词 主动防御; 网络欺骗; 动态指纹; 数据流匿名

中图法分类号 TN 915.08 DOI 号 10.19363/J.cnki.cn10-1380/tn.2022.03.05

An Anonymous Network Deception Method Based on Container Fingerprint Modification for SaaS Applications

LI Lingshu[1*], WU Jiangxing[1], LIU Wenyan[1]

[1]National Digital Program-Controlled Switching center People's Liberation Army Strategic Support Force Information Engineering University, Zhengzhou 450001, China

Abstract With the popularization of cloud computing technology in key industries, Software-as-a-Service (SaaS) services are widely used in industries. The security problem of the cloud environment is becoming increasingly prominent. As locating the target cloud resources is the pre-step of a network attack, network deception technology can effectively disrupt the attackers' network reconnaissance and hide the fingerprint information of important network assets (service port, operating system type, etc.). However, due to the relation between the fake fingerprint and the real device, an advanced attacker can locate the target device through multi-dimensional information. The principle of the deception method based on container fingerprint anonymity is hiding important sensitive services in a large number of ordinary non-sensitive services. Multiple deception techniques are used to counter network reconnaissance. The anonymity algorithm of batch processing will lose the fast and convenient characteristics of cloud services. Cloud services need to adopt a real-time and strong continuous anonymity method. In order to prevent essential cloud resources from locating and tracking, this paper proposes a network deception method based on container fingerprints anonymity. By modifying the containers' fingerprints in the cloud resource pool, false cloud resource views are created to confuse the attacker and improve the difficulty of network detection. In order to reduce the overhead of container fingerprint modification and possible service delay, a category attribute measurement method based on semantic level is proposed as the optimization goal of the container fingerprint anonymity algorithm. Since the container fingerprint information that needs to be modified is a continuously generated data stream, a dynamic fingerprint spoofing algorithm CFDAA based on data stream anonymity is proposed for real-time online processing. It realizes the rapid container fingerprint modification through delay control and cluster segmentation. Moreover, k-anonymity and l-diversity are satisfied. Experimental results show that the proposed method can effectively improve attackers' difficulty of network reconnaissance with controllable overhead.

Key words active defense; network deception; dynamic fingerprint; data stream anonymity

通讯作者: 李凌书, 博士生, Email:lls.ndsc@aliyun.com。

本课题得到国家自然科学基金项目(No. 62002383)、国家重点研发计划课题(No. 2018YFB0804004)资助。

收稿日期: 2020-11-27; 修改日期: 2021-02-27; 定稿日期: 2022-01-20

1 引言

网络杀伤链(Cyber Kill Chain, CKC)模型将网络入侵划分为多个阶段, 其中网络侦查是其中的第一个步骤, 包括攻击目标的识别、选择和分析, 通常使用网络爬虫、流量分析等技术获取目标分析[1]。网络侦查、网络嗅探作为网络攻击的前置步骤, 对网络系统带来巨大的潜在威胁。如在软件即服务(Software as a Service, SaaS)云中, 云服务提供商(Cloud Service Provider, CSP)业务应用的部署模式和策略属于商业机密, 攻击者一但获取就可以进一步展开针对性的攻击。研究表明网络攻击者 95%的时间用于网络嗅探, 实施具体网络攻击的时间只占 5%[2]。

云架构下的业务功能面临严重的来自嗅探攻击的威胁, 虚拟化的重要资源和服务更容易被攻击者定位, 主要有以下三点原因: (1)云环境缺乏物理边界。在传统企业网络或者 ICT 基础设施中, 网络隔离是一种常用的防嗅探机制, 如采用内外网隔离模式, 将核心服务器放置在内网中进行保护。而云环境由于虚拟化、分布式部署、远程接入等特点, 物理边界模糊, 为嗅探攻击创造了有利条件[3]。(2)逻辑边界不可靠。云环境中一般对虚拟机和容器进行隔离, 创建可信子空间, 如虚拟子网、网络切片、微隔离等技术。然而攻击者可通过诸如挂马攻击、攻击虚拟化层等方式突破隔离技术, 因此很难保证逻辑边界可靠和可信子空间完全可信。(3)云系统的确定性和同构性。攻击者可以对目标云资源进行反复嗅探和渗透, 不断积累攻击经验, 而防御者却不能借此改善防御措施, 造成攻防不对称。鉴于以上分析, 提高云环境对于网络嗅探的抗攻击性, 破坏或干扰攻击者的前期信息收集, 阻断或误导后续攻击, 具有十分重要的研究意义。

网络欺骗(Cyber Deception, CD)技术作为一种网络主动防御技术, 试图改变传统网络系统的确定性、静态性和同构性。网络欺骗技术旨在通过制造虚假信息(虚假、混淆、误导性的响应)来干扰攻击者的认知过程, 扰乱攻击者的自动化工具。根据欺骗信息构建方式的不同, 网络欺骗可以分为“掩饰”和“模拟”两种[4]。“掩饰”是对真实目标进行伪装, 通过掩盖其特征信息来避免被攻击者定位; “模拟”是对虚假目标进行伪装, 通过构建虚假的主机或服务引导攻击者去攻击错误的对象。我们将可以直接或间接标识出特定设备的特征属性或标识信息称为网络指纹[5]。网络指纹包括应用层指纹、数据层指纹、网络层指纹、设备层指纹等。指纹探测攻击指利用 IP、ICMP、TCP 和 UDP 等协议, 对云中的容器或虚拟机进行操作系统、镜像版本、应用类型等进行探测, 为发起后续攻击提供基础。SaaS 云中对探测攻击的网络欺骗技术可以通过操纵容器指纹来实现, 如通过操作系统混淆技术[6]、虚拟地址技术[7]、虚假 FTP 服务模拟[8]或地址随机化技术[9]实现“掩饰”欺骗, 或通过蜜罐、蜜网等欺骗技术实现“模拟”欺骗[10]。

虽然网络欺骗的研究涌现出诸多成果和工具, 但现有网络欺骗方法依然存在一些缺陷和不足。(1)安全性不足。随着攻击技术的复杂化、持久化, 一些研究表明攻击者往往会结合目标设备的多维指纹信息实现身份定位, 一些关键特征属性值的组合有可能标识某些功能或者服务[11]。如即使隐匿了 IP 信息, 攻击者依然可以通过结合操作系统类型和端口等信息找到容器和服务的对应关系。(2)开销大。部署影子主机或者高交互的蜜罐消耗大量的资源, 部分研究方案中正常主机和蜜罐等防御主机的数量比为 1∶n[12]。(3)处理效率低。容器云中网络资源具有快速弹性伸缩、按需配置的特点, 容器池中容器的数量和类型随着业务负载不断变化, 容器指纹信息是一个持续高速产生的动态数据流, 现有欺骗算法难以对其进行处理[13]。

针对上述三个问题, 本文提出一种 SaaS 云环境下基于容器指纹匿名的网络欺骗方法。首先, 为保护重要的容器, 管理员配置其他非重要容器、影子容器或蜜罐使其呈现相同的多维网络指纹信息, 进而增大攻击者定位目标主机的难度, 我们将其称为“容器指纹匿名”。其次, 鉴于 SaaS 云中的针对某种特定服务的攻击者仅关心与之相关的特定容器。例如, 攻击者意图对某金融业务的容器进行信息窃取, 则承载其他业务(如视频播放)的容器的攻击价值很低, 甚至不具有攻击价值。据此, 所提指纹匿名欺骗方法将多个正常的容器和蜜罐容器组成一个簇, 相当于多个正常容器共享蜜罐带来的安全增益, 使得正常主机和蜜罐等防御主机的数量比为 m∶n, 减少蜜罐等防御容器的比例, 降低资源开销。最后, 针对容器指纹持续产生的特点, 提出一种基于时延约束和簇分割的容器指纹信息流式处理算法 CFDAA(Container Fingerprint Dataflow Anonymous Algorithm), 在满足实时性约束的条件下, 输出开销最小的容器指纹修改方案。实验结果表明, 所提算法能够在开销可控的情况下, 有效提高攻击者定位目标云资源的难度。

本文后续部分安排如下。第 2 章梳理了相关背景及研究现状, 首先通过将网络欺骗技术与传统安全技术进行对比, 厘清与移动目标防御等技术的区

别与联系, 然后介绍欺骗技术的研究现状; 第 3 章主要首先介绍所提欺骗方法的原理, 然后介绍实现框架, 最后对其安全性进行分析; 第 4 章分析容器指纹信息高速动态产生的特点, 提出面向数据流的匿名方法, 并提出指纹修改开销的衡量方法; 第 5 章具体介绍算法的流程和实现, 并对复杂度进行分析; 第 6 章通过实验验证所提方法的有效性, 并从安全性和性能两方面进行分析; 第 7 章对全文进行总结, 并提出未来的研究方向。

2 背景及研究现状

在信息安全领域, 网络欺骗技术可以追溯到 20 世纪 80 年代[14]。防火墙、身份验证等基于边界的安全防御技术难以应对内部攻击和社会工程学攻击, 而诸如基于特征的威胁检测系统等深度防御(Defense-In-Depth, DID)技术, 存在误报率高的问题。蜜罐类技术由于其低误报和一定的 0-day 漏洞检测能力, 引起了学术界和产业界的广泛重视, 并提出了网络欺骗的概念。21 世纪初由于蠕虫等恶意代码泛滥, 蜜罐技术得到进一步地广泛研究[15]。2011 年后, 随着应用场景的演进和高级持续性威胁(Advanced Persistent Threat, APT)的出现, 网络欺骗的内涵和外延也不断扩大, 不再局限于蜜罐类技术, 逐渐发展为涵盖网络流量仿真、系统混淆、地址随机化等多种手段的主动防御技术。

本节首先阐述网络欺骗技术的基本概念, 并与传统 IT 安全技术进行对比, 最后简述网络欺骗技术的研究现状。

2.1 基本概念

网络欺骗技术旨在误导、扰乱攻击者而采取的有计划的行动, 诱使攻击者采取或不采取特定的动作来增加系统安全性[16]。与传统关注机密性、完整性、可鉴别性和可用性(Confidentiality Integrity Authentication Availability, CIAA)的 IT 安全技术相比[17], 网络欺骗技术的技术思路有一定的区别, 追求入侵检测或是捕捉攻击者的行为。网络欺骗主要关注机密性、可鉴别性、可控性和可用性(Confidentiality Authentication Controllability Availability, CACA)[4]。通过对比可以看出欺骗技术不侧重完整性保护问题, 而是关注自身系统的可控性, 不能因为引入欺骗措施而带来额外的安全隐患, 自身安全可控是捕捉和分析攻击行为的前提。网络欺骗技术有助于发现网络攻击, 消耗攻击者的时间、精力与资源, 威慑网络攻击, 为攻击溯源等后续应对措施提供数据和信息[5]。

类似 Fraunholz 等人[17]的归类方式, 我们将网络欺骗技术与传统 IT 安全技术的关系进行总结, 如图 1 所示。图中白色框中的技术为传统 IT 安全技术, 灰色框中的技术为网络欺骗技术。横坐标为网络安全防御的三个阶段: 威胁阻止(Threat Prevention)、威胁检测(Threat Detection)和威胁响应(Threat Correction)。纵坐标根据网络欺骗在信息系统的作用点不同, 分为网络层、系统层、数据与应用层。

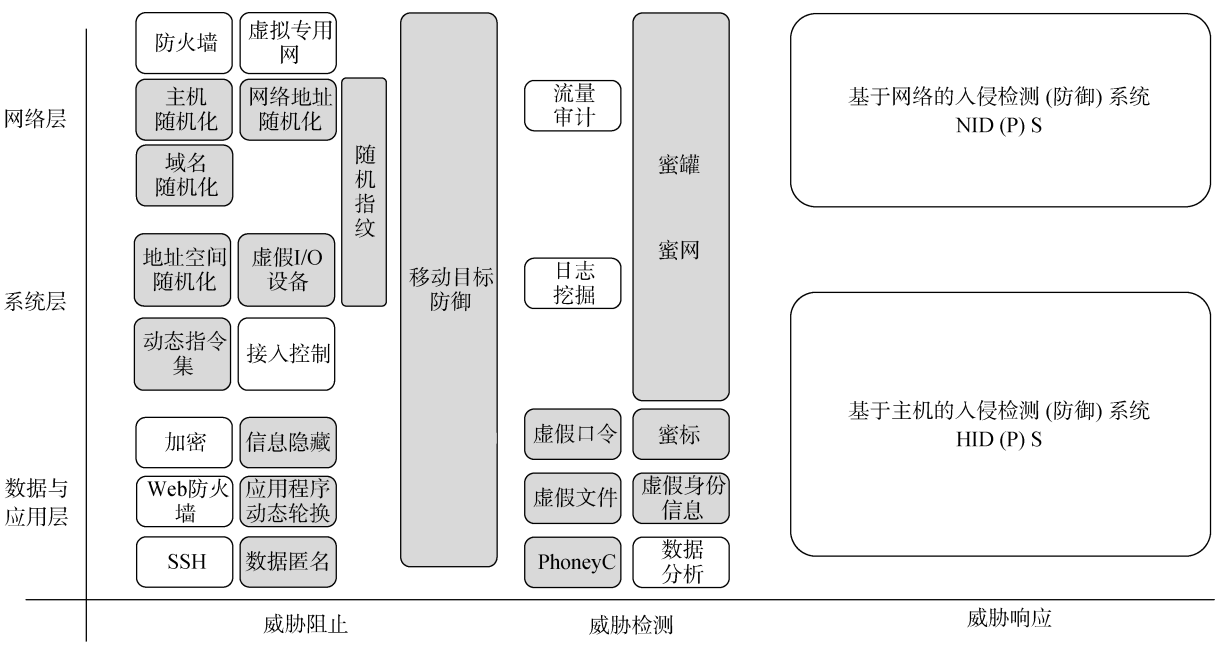

图 1 网络欺骗技术与传统 IT 安全技术的关系图

Figure 1 Overview of cyber deception technique and relevant security techniques

在威胁阻止方面, 网络欺骗技术主要是借助一些移动目标防御(Moving Target Defense, MTD)技术制造信息系统的不确定性, 如网络地址随机化(Network Address Space Randomization, NASR)[18]、随机指纹(Random Fingerprinting, RF)[19]、动态指令集等技术[20]。在威胁检测方面, 网络欺骗主要以蜜罐蜜标类技术为主, 思路上类似于一个基于异常的入侵检测系统(Intrusion Detection System, IDS)。可以看出网络欺骗是一种新型的网络主动防御方法, 不再局限于蜜罐技术, 与移动目标防御技术的范围有一定的重叠。此外, 网络欺骗技术可以与传统的安全技术叠加使用, 以获得更好的安全效果。

2.2 研究现状

网络欺骗技术范围广泛, 本文主要关注以指纹修改技术、蜜罐类技术、MTD 技术为核心的相关欺骗技术。

2.2.1 指纹修改技术

网络指纹是指直接或间接标识出特定设备的特征属性或标识信息, 攻击者可以通过网络嗅探获取设备的部分指纹信息[21]。攻击者可以利用 Nmap 设计畸形的 TCP 和 UDP 探测包, 根据响应数据分析协议序号(ISN)的生成规则、系统的时钟情况等信息, 识别出目标设备的操作系统。随着海量物联网设备涌入网络空间, 设备指纹被广泛应用于设备识别和设备定位[5]。进而, 攻击者可以通过网络指纹对目标设备进行定位和追踪。如对 web 用户进行长期跟踪的攻击者可以利用操作系统、分辨率、像素比等浏览器指纹特征, 即使攻击者定期清理浏览器 Cookie, 也可以对目标进行定位[22]。

为抵御网络嗅探与定位, 指纹修改、指纹随机化技术通过主动混淆和篡改指纹, 对攻击链进行置乱。如 TCP/IP 协议栈伪装工具通过修改协议栈的某些参数使得对某些数据包的回应数据包携带与真实指纹信息不同的特征, 从而欺骗攻击者。在网络层基于软件定义网络(Software Defined Network, SDN)的指纹修改欺骗可以通过 IDS 和指纹修改引擎配合实现[23-24]。指纹修改引擎可以直接操纵因特网控制消息协议(Internet Control Message Protocol, ICMP)消息来对流量进行刻意混淆, 但这种主动的置乱也可能导致未知的服务终端与失效。Ip Personality[6]、Fingerprint Fucker[25]等工具参照真实的操作系统指纹特征来修改当前 TCP/IP 协议栈中的网络特征值, 从而达到操作系统伪装的目的。Rahman 等人[26]通过对数据包进行混淆隐藏设备指纹, 鉴于指纹欺骗会造成明显的性能下降, 故提出一种基于信号博弈论的指纹动态修改机制。

2.2.2 虚拟蜜罐类技术

以资源类型作为分类标准, 可将蜜罐类技术分为实物蜜罐、虚拟蜜罐和半实物蜜罐[10]。虚拟蜜罐广泛应用于云计算环境下的网络安全领域研究。

虚拟蜜罐是在虚拟化技术的基础上使用软件实现的蜜罐, 随着虚拟机、容器技术的发展, 虚拟蜜罐具有部署灵活、开销小等优点。虚拟蜜罐针对特定的诱捕需求进行设计, 模拟目标操作系统和业务, 诱使攻击者进入防御者可以部署的隔离的虚拟环境, 并与攻击者保持交互以使攻击者暴露更多的威胁情报[27-28]。经过近二十年的发展, 诸如 ScriptGen[29]等虚拟蜜罐自动脚本生成工具已较为成熟。虽然基于软件实现的虚拟蜜罐与硬件解耦, 部署配置更加方便, 但基于虚拟化技术进行模拟的仿真程度与真实环境仍存在差距。

虚拟蜜罐具有更强的灵活性和动态性。Fan 等人[30]提出一个通用的虚拟蜜网管理架构, 其基于对请求的语法规范检查生成蜜网模板, 实现蜜网的自动化生成。基于该架构, 该团队随后提出一种虚拟蜜网自适应部署系统, 可以动态创建、配置与部署高低交互蜜罐来模拟多个操作系统[31]。Fraunholz 等人[32]基于机器学习技术实现虚拟蜜罐的配置, 提出一种基于聚类的网络设备识别机制, 然后将蜜罐和真实主机一起部署, 从而以最少的配置开销获取最多的威胁信息。

2.2.3 基于 MTD 的欺骗技术

MTD 与网络欺骗在思想上有一定的共通性。为了降低网络侦察的有效性, Sun 等人[33]提出 IP 随机化技术与网络欺骗技术相结合的防御机制, 该机制可以动态产生大量虚假的网络拓扑, 从而使攻击者从网络扫描中获得的知识无效。Jafarian 等人[9]提出一种基于时空随机化的 IP 跳变机制, 这种 MTD 策略迫使攻击者频繁地重新扫描网络, 以减慢攻击进度, 提高攻击的门槛。Sharma 等人[7]提出了基于 SDN 网络的虚拟 IP 随机复用技术, 真实 IP 通过复用多个随时间变化的虚拟 IP, 以降低攻击者探测到真实系统 IP 指纹的概率。

为应对高级攻击者, 研究者开始将多种欺骗策略融合使用。Duan 等[12]提出一种网络欺骗框架, 将动态轮换、多样化、蜜罐技术结合来提高网络欺骗的能力, 并评估了不同参数对欺骗防御效果的影响。但其匿名策略简单, 没有考虑动态数据导致的信息泄露问题。Jafarian 等人[11]通过 IP/MAC 地址、开放/关闭端口、服务名称、可利用漏洞等来描述网络主

机提出一种匿名化的主动防御系统 HIDE, 通过符合匿名规则的方式同时持续变换 IP 地址和服务器指纹, 让攻击者无法区分蜜罐和真实主机。

通过回顾和分析上述研究文献, 可以发现现有 MTD 类欺骗技术的普遍存在频繁的动态变化与清洗, 造成服务延迟甚至中断; 而蜜罐类欺骗技术部署虚假主机或服务消耗大量资源, 尤其是高交互型蜜罐。因此本文提出一种 SaaS 云中基于容器指纹匿名的网络欺骗方法, 通过修改云资源池中容器的指纹满足匿名化标准来提高安全性, 减少盲目的跳变和蜜罐部署。

3 原理与框架

本节首先介绍 SaaS 云环境中容器指纹匿名欺骗的原理, 并介绍其实现方式, 然后建模对其安全性进行分析, 并对关键概念进行定义。

3.1 指纹匿名欺骗原理

诸多指纹修改技术[6, 34]可以实现各个层面的不同指纹的混淆和伪装。然而单一属性的指纹修改难以欺骗高级攻击者, 若攻击者拥有的知识或相关信息能够帮助其推理和获取容器的隐私属性, 如服务端口与服务类型的对应关系, 则可以发动背景知识攻击(Background Knowledge Attack, BKA)。一些研究已经开始探索多属性的跳变, 如 SDN 中基于时间和空间两个维度的 IP 跳变技术[34]和基于 IP、MAC 协同跳变的技术[18]。然而即使不考虑性能和开销问题, 无论指纹修改多么频繁, 依然存在一个虚拟指纹和真实指纹的对应关系, 一旦这种对应关系被攻击者掌握, 攻击依然是可达的。

基于容器指纹匿名的欺骗方法的原理是试图将重要的敏感容器隐藏到大量普通的非敏感容器中。通过指纹修改技术使得云资源池中的容器满足 k-匿名化要去, 即对于每个容器, 在同一个网络中都存在一组具有相同指纹的 k–1 个其他容器。云环境下指纹匿名欺骗原理如图 2 所示。容器指纹修改通过欺骗网关实现消息修改和地址/端口跳转。每个容器上承载着一个服务, 如 MySQL 或 Web。容器的真实指纹可能经过修改之后形成虚假指纹, 暴露给攻击者。指纹是多个属性的集合, 如操作系统类型、容器类型等。

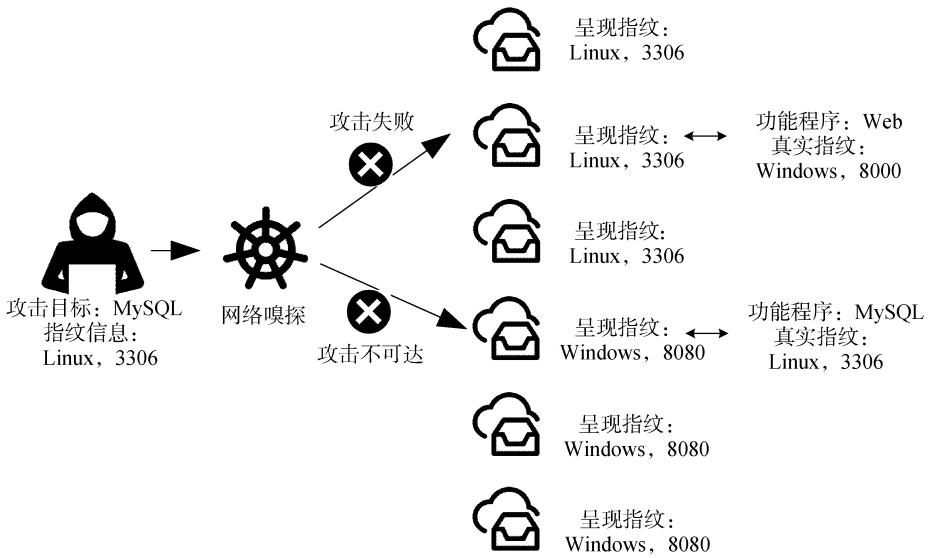

图 2 云环境下指纹匿名欺骗原理示意图

Figure 2 Fingerprint anonymous deception in cloud environment

攻击者先通过网络嗅探进行目标容器定位, 再进行后续的攻击。假设攻击者的攻击目标是 MySQL 数据库, 攻击者的背景知识是容器为 Linux 系统, 端口号为 3306。由于采用指纹修改技术对运行 MySQL 的容器进行了伪装, 攻击者可能无法发现目标容器, 导致攻击不可达。即使攻击者识破了指纹欺骗, 获得了运行 MySQL 的容器指纹为{Windows, 8000}, 由于 k 个容器对外呈现相同的指纹, 攻击者可能攻击到错误的目标而导致攻击失败, 增大了攻击者的攻击难度。为进一步增加系统安全性, k 个相同指纹容器中还可以部署部分虚拟蜜罐, 正常用户一般不会对其进行交互。一旦攻击者访问到蜜罐容器, 则触发安全告警, 对攻击者进行分析和溯源。

3.2 实现框架

云数据中心部署容器指纹欺骗的实现框架如图 3 所示, 主要包括业务部署和用户请求两个流程

的实现。对于业务部署流程，容器云编排管理器 Kubernetes 根据业务请求实例化新的容器。欺骗网关接收到这些新容器的指纹信息，结合预先输入的地址空间、容器配置(操作系统、服务类型、漏洞)、欺骗成本、预算限制、容器价值、风险约束、可行性约束(指纹是否支持修改)等进行综合分析。如果存在满足约束条件的可行解，欺骗引擎执行欺骗策略；若不满足，则适当增加预算或放松风险约束。如果欺骗策略需要创建蜜罐容器，则将需要实例化的蜜罐容器配置参数返回给 Kubernetes，Kubernetes 接收到蜜罐生成消息后按照配置参数生成相应的蜜罐。

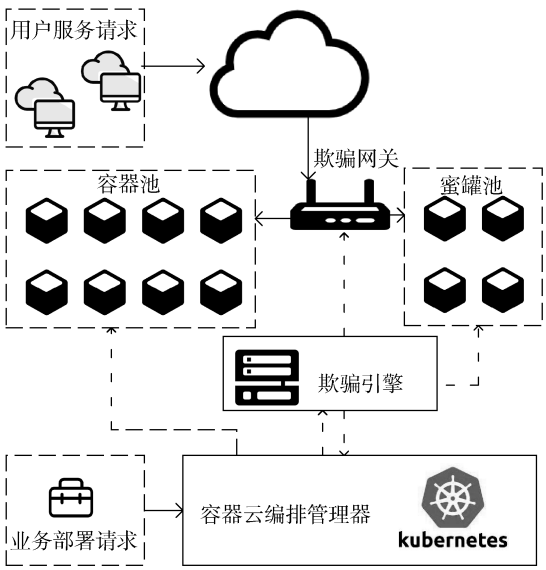

图 3　SaaS 云中容器指纹欺骗的实现框架
Figure 3　Containers fingerprint deception implementation framework in the SaaS cloud

对于用户访问流程。用户客户端的请求经过云网络到达欺骗网关，通过欺骗网关与目标网络进行代理通信。欺骗引擎负责确定容器的伪装方式，并将其下发给欺骗网关。欺骗网关是位于虚拟子网边界的分布式设备。欺骗网关的功能类似一个网络地址和端口转换(Network Address Port Translation, NAPT)设备，根据从欺骗引擎接收到的欺骗策略，实现地址/端口跳变，将流转发到未使用端口，或者转发到蜜罐容器中的蜜罐服务。

欺骗技术可以与其他安全技术协同部署，以取得超线性的安全增益。因此欺骗网关还具有虚拟地址跳变功能，增加云资源池拓扑的不确定性。此外，由于攻击者可以通过分析网络流量所属的应用类型(如邮件类、多媒体类、网站类等)，进而针对性的进行流量拦截和指纹攻击。因此，欺骗网关生成的欺骗流量还需要能够对抗基于神经网络等智能模型的网络探测与侦查[35-36]。

具体实现上，本文提出的欺骗网关主要完成 IP 包头、TCP 包头以及消息响应时间的修改，我们使用数据平面开发套件(Data Plane Development Kit, DPDK)部署指纹欺骗代理网关，DPDK 作为高性能的数据包处理工具，为用户提供了方便的数据包处理函数，数据包的处理绕过了内核，能够在用户态以轮询的方式完成数据包的处理，极大的提高了数据的处理能力以及吞吐量，此外，目前主流的云提供商如阿里云、腾讯云、京东云均采用 DPDK 作为云接入网关，所以采用 DPDK 部署指纹欺骗代理网关具有部署便捷和高可用的优势。当采用 DPDK 部署欺骗网关时，DPDK 以轮询的方式从网卡接收到来自客户端和服务端的报文，当接收到来自客户端的数据包时，在完成数据包源 IP 和源端口号的修改之后，DPDK 将数据包发送到服务端；当接收到来自服务端的数据包时，在完成数据包目的 IP 和目的端口号的修改之后，DPDK 将数据包发送到客户端。整个过程完全在用户态实现，避免了因报文从内核态到用户态的拷贝带来的中断开销，增加了报文的处理能力。

3.3　安全性分析

容器指纹匿名欺骗框架的理论安全增益来源于 *k*-匿名技术，*k*-匿名由 Samarati 和 Sweeney 提出[37]。*k*-匿名的基本原理是通过一定的方法将数据表中的元组进行分类，并进行匿名处理，使其形成至少含有 k 个除敏感属性值外不可区分元组的若干等价类，对未包含在等价类内的剩余的元组进行隐匿或泛化到最高层。通过 *k*-匿名技术，使得攻击者识别用户身份的概率至多为 $1/k$。随着数据挖掘等技术日益受到关注，如何保护敏感属性方面的研究已有诸多成果[38]。

下面介绍本文中使用的一些概念术语和数学符号。假设 $T(A_1, A_2, ..., A_n)$ 是包含有限数量容器指纹属性的一个原始数据表(Original Data Table, ODT)，其中 A_i 为容器的一项指纹属性。$t(A_1, ..., A_n)$ 表示 T 中的一个有序序列，即一个容器的所有指纹属性。$T[A_1, ..., A_m]$ 表示属性 $A_1, ..., A_m$ 的投影。

定义 1　敏感属性 (Sensitive-Identifier, SI)

在指纹匿名欺骗模型中，敏感属性是防御者不想暴露的隐私属性，如容器的用户归属，承载的服务类型等。

定义 2　准标识符属性 (Quasi-Identifier, QI)

准标识符属性不能唯一标识容器敏感属性，但

与其他标识符属性联合可能确定出容器的敏感属性, 用 A^{QI} 表示。容器的每个指纹属性都可以作为准标识符, 一般根据专家经验或基于对攻击者的手段和工具的知识来进行准标识符的选取。

定义 3　等价类(Equivalence Class, EC)

原始数据表 T 在属性集 $\{A_1,...,A_m\}$ 上的等价类指的是数据表 T 中在 $\{A_1,...,A_m\}$ 上的属性值完全相同的所有元组的集合。即对于元组集合 $\{t_1,...,t_m\}\subseteq T$, $\forall t_a,t_b\in\{t_1,...,t_m\}$, 有 $t_a[A_i,...,A_j]=t_b[A_i,...,A_j]=(a_i,...,a_j)$; 且对于 $t_c\in T-\{t_1,...,t_m\}$, 有 $t_c[A_i,...,A_j]\neq(a_i,...,a_j)$ 。

定义 4　泛化(Generalization)

泛化又称概化, 是匿名技术的典型方法, 其通过扩大准标识属性描述范围, 将多个准标识属性融合为一个准标识属性, 达到降低通过准标识符推断隐私属性的概率。本文中引入泛化的概念主要用于指纹修改开销的计算。

定义 5　*k*-匿名

若所有等价类均至少含有 k 个 QI 值不可区分的元组, 则该数据集实现了 *k*-匿名化。将匿名数据表(Anonymous Data Table, ADT)记为 T^* 。*k*-匿名的过程即表示为:

$$T=\{A^I,A^{QI},A^S\}\rightarrow T^*=\{A^{QI},A^S\} \tag{1}$$

下面通过一个实例对容器指纹匿名进行说明。假设一组 SaaS 业务的容器指纹信息如表 1 所示, 该组上线的容器个数为 6, 其中敏感属性为应用程序类型, 准标识属性是 IP 地址、操作系统、容器类型和镜像版本。容器上线前要对其指纹进行修改以满足匿名化要求, 即基于表 1 通过匿名化推导得出一个满足 3-匿名要求的匿名数据集, 如表 2 所示。每行记录在类标识属性集上的投影值行都有与之相等其他至少 2 个值行。例如, 对于第 3 个记录, 存在第 4 个记录与之在准标识属性集上的投影值相同。可以看出, 若随着 k 值的增大, 等价类大小会增加, 容器指纹修改幅度会增大, 而安全性将提升。

表 1　一组 SaaS 服务的容器指纹数据

Table 1　A set of container fingerprint data for SaaS services

业务编号	操作系统	镜像版本	服务端口	应用程序
业务 1	Redhat 7	Jboss 5.0	8080	Apache
业务 2	CoreOS	Tomcat 7	80	Traefik
业务 3	Windows 2016	Tomcat 8	2379	ETCD
业务 4	Redhat 7	Tomcat 8	3306	MySQL
业务 5	Ubuntu 12	Tomcat 7	2379	ETCD
业务 6	CoreOS	Tomcat 8	3306	Apache

表 2　表 1 的一个 3-匿名导出表

Table 2　A 3-anonymous export table from table 1

业务类型	操作系统	镜像版本	服务端口	应用程序
业务 1	Redhat 7	Jboss 5.0	2379	Apache
业务 2	Redhat 7	Jboss 5.0	2379	Traefik
业务 3	Redhat 7	Jboss 5.0	2379	ETCD
业务 4	CoreOS	Tomcat 8	3306	MySQL
业务 5	CoreOS	Tomcat 8	3306	ETCD
业务 6	CoreOS	Tomcat 8	3306	Apache

然而, *k*-匿名模型虽能有效避免重要的业务服务被嗅探攻击直接识别, 让攻击者难以定位承载目标业务的容器, 但由于其未对指纹相同的容器所承载的业务类型分布进行约束, 当等价类中敏感值语义相似时, 易受到偏斜型攻击。假设攻击者知道一种重要业务需要使用 Windows 操作系统和 Etcd 数据库, 推测表 2 等价类 2 中的某一个元组为其容器指纹, 虽然等价类 2 中元组的敏感属性值各不相同, 但都是数据库业务, 尽管无法得知具体是哪种数据库, 但仍发生了隐私泄露。为解决匿名模型不能抵制偏斜型性攻击和背景知识攻击的问题, 提出多样性模型, 其主要思想是要求每个等价类中敏感属性值都必须包含 l 个不同的元素, 考虑了对敏感属性的约束。

4　基于容器指纹匿名的网络欺骗方法

基于一个原始多属性数据集导出最优的 *k*-匿名数据集是一个 NP 难问题, 而真实场景中容器指纹匿名问题更加复杂。SaaS 云中网络资源具有快速弹性伸缩、按需配置的特点, 容器池中容器的数量和类型随着业务负载不断变化, 欺骗引擎需要对一个持续产生的动态数据流进行匿名。

4.1　数据流匿名

将云资源池中的设备指纹数据建模为一个无限增减的包含时间戳的数据流, 其具有数据量大、连续到达、存在先后顺序的特点。对于数据流 S 中的一个元组 s, 其时间戳用 $s.t$ 表示。

定义 6　流数据 *k*-匿名

对于数据流 $S=\{s_1,s_2,...,s_n\}$, 存在输出数据流 S_{out} 使得对于每一个 $s\in S$ 都有 $s'\in S_{out}$, 且对于任意一个 $s'\in S_{out}$, 其从属的等价类为 E , $DP(E)$ 为等价类 E 对应的彼此相异的原始容器指纹元组集。若对于任意 E , 都有 $|DP(E)|\geqslant k$, 则 $S\rightarrow S_{out}$ 称为数据流 *k*-匿名。

新实例化的容器要及时上线, 因此流数据 *k*-匿名算法的及时性非常重要, 我们将一个容器从准备上线

到进行指纹匿名化的最大等待时间定义为时延约束。

定义 7　时延约束 δ

设有数据流 k-匿名方法 F ,若 F 输出的 k-匿名数据流 S_{out} 满足 $\forall s' \in S_{out}$, $s'.t - s.t < \delta$, 其中 s 为原数据流 S 中与 s' 对应的元组, δ 为给定正整数, 则称 F 满足时延约束 δ 。

定义 8　抑制 (Suppression)

抑制是指将等价类 E 中所有容器指纹元组的某一个具体的属性 A_i 替换成相同值, 使该等价类中所有元组在该属性上的值都相同。例如, 将所有服务端口修改为某固定端口。

4.2　指纹修改开销

容器指纹匿名技术基于指纹修改技术, 而容器指纹修改带来计算开销的同时还可能导致服务延时甚至中断, 因此匿名算法应尽量减少指纹修改的程度, 且避免对已发布匿名化指纹进行二次修改。为衡量匿名算法的开销, 首先需要对容器指纹修改开销进行量化, 通常采用两个指纹矢量的距离来衡量开销。鉴于容器指纹元组 s 中的属性 A_i 均为范畴属性, 即具有一定语义的非数值型属性, 一种常用的距离定义方法为: 若两个属性值相同, 则距离为 0, 否则距离为 1。但容器指纹的属性值之间存在一定的语义关系, 如同样是操作系统指纹, 从 Ubuntu 修改到 Debian 的开销要小于修改到 Windows, 因此这种距离定义方式过于粗糙。故本文根据自然语义关系构造出分类树, 范畴属性值则对应于分类树的叶子节点, 非叶子节点则是对子树范围内属性值的泛化。两个属性值之间的距离定义为它们具有最小公共祖先的子树高度和整个分类树高度之比, 从而避免了范畴属性不能对其进行距离度量的问题。在此基础上, 给出了一种基于语义等级的范畴属性度量方法, 不仅考虑泛化树中公共祖先子树的高度问题, 还将范畴属性(叶子节点)之间的语义距离计算在内, 从而准确的表示了距离, 聚类效果也变得更加的准确。以 OS 属性为例, 容器指纹数据集 OS 属性的语义分类树如图 4 所示。

对于准标识属性 $q^i \in A^{\mathrm{QI}}$, DGH_i 为属性值 q^i 的语义分类树。对于 DGH_i 中的一个叶节点 v , 将其指纹修改到叶节点 u , 其指纹修改开销指数定义为

$$d(v,u) = \frac{|D_{v,u}| - 1}{|D| - 1} \tag{2}$$

其中 $|D|$ 为 DGH_i 中所有的叶节点的集合, $|D_{v,u}|$ 为 DGH_i 中以节点 v 和节点 u 最近公共祖先为根节点的子树的所有叶节点的集合。例如在图 4 的语义分类树中, 将指纹属性从 Ubuntu 修改到 Centos,

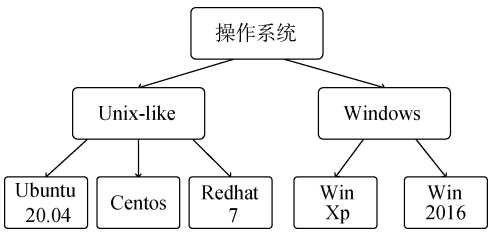

图 4　容器指纹数据集 OS 属性的语义分类树

Figure 4　The semantic classification tree for OS attributes of the container fingerprint dataset

其最近的公共祖先为 Unix-like, 因此指纹修改开销为 $d(Ubuntu, Centos) = \dfrac{4-1}{8-1} = \dfrac{3}{7}$ 。值得一提的是, 在如果要将OS属性修改到Unix-like这一级, 是将所有的需要修改的属性值修改为某一个叶节点 $v \in S_{\mathrm{Unix-like}}$ 的属性值。

若 $f_i(a_1^i, ..., a_n^i)$ 表示容器 s^i 的 n 维指纹。则指纹从 s^x 修改为 s^y 需要付出的开销定义为:

$$\gamma(s^x, s^y) = \frac{1}{n}\sum_{i=1}^{n} \varpi_i d(a_i^x, a_i^y) \tag{3}$$

其中 a_i^x 表示容器 s^x 的第 i 个指纹, $d(a_i^x, a_i^y)$ 表示将 a_i^x 修改为 a_i^y 的开销, ϖ_i 为第 i 个指纹修改困难程度的权重。将容器 s^x 的指纹修改为 s^y , 需要对所有的 n 个指纹进行修改。

对于一个容器等价类 $\{s^1, s^2, ..., s^m\}$, 指纹修改的欺骗开销定义为:

$$\gamma(s^1, s^2, ..., s^m) = \min_{j \in [1,...,m]} \sum_{i=1}^{m} \gamma(s^i, s^j) \tag{4}$$

以上开销计算方式对所有属性信息都一视同仁, 而指纹匿名问题中不同属性值的修改难度是不同的。本文引入开销权重矩阵, 其为 $m \times (n+1)$维矩阵, 其中 m 表示敏感属性值的数量, n 表示准标识符属性的数量。则该矩阵可表示为:

$$M \mid S = (t_{ij} \mid b_i)_{m \times (n+1)} = \left[\begin{array}{ccccc|c} QI_1 & QI_1 & QI_1 & \cdots & QI_1 & SI_1 \\ \lambda_{11} & \lambda_{12} & \lambda_{13} & \cdots & \lambda_{1n} & b_1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \lambda_{i1} & \lambda_{i2} & \lambda_{i3} & \cdots & \lambda_{in} & b_i \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \lambda_{m1} & \lambda_{m2} & \lambda_{m3} & \cdots & \lambda_{mn} & b_n \end{array}\right] \tag{5}$$

由此, 一个容器等价类 $\{s^1,s^2,...,s^m\}$ 的指纹修改欺骗开销定义为:

$$\gamma(s^1,s^2,...,s^m)=\min_{j\in[1,...,m]}\sum_{i=1}^{m}\lambda_{ij}\gamma(s^i,s^j) \qquad (6)$$

5 基于聚类的指纹匿名欺骗算法

针对容器指纹持续产生的特点, 本节首先介绍算法的设计思想, 然后提出一种基于时延约束和簇分割的容器指纹信息流式处理算法, 阐述算法实现细节, 最后对算法复杂度进行分析。

5.1 算法设计思想

为保证对实时生成上线的容器进行指纹匿名, 本节提出一种基于时延约束和簇分割的指纹匿名欺骗算法 CFDAA。由于新容器上线具有不确定性, 若是让容器在队列中等待, 等到有足够数量容器后再批次处理, 会损失云服务快速便捷的特性。因此容器匿名欺骗需要采用一种实时强的连续匿名方法, CFDAA 引入时延约束, 强制对到达时间超过 δ 的容器进行匿名处理。

CFDAA 的主要设计思想为:

(1) 通过从数据流中生成簇结构, 然后基于簇发布满足匿名要求的等价类, 从而对容器指纹信息数据流进行保护。在整个隐私保护发布过程维护一个候选 k-匿名簇的集合 Γ 和当前云资源池已发布 k-匿名簇的集合 Ψ。一个 k-匿名簇即对应于一个容器指纹的等价类。

(2) 对数据流 S 中任意时刻到来的元组 s, 首先根据元组与元组之间或元组与聚类簇之间的加权距离和信息损失大小, 为其从 Γ 中选择一个簇, 或新创建一个簇并加入 Γ 中。

(3) 为提高抵抗偏斜型攻击等威胁的能力, 要求生成的匿名簇满足 l-多样性。当匿名簇不满足 l-多样性时, 可以进行簇合并以增加敏感属性种类, 但会导致指纹修改开销增大。由于蜜罐可视为拥有彼此相异的敏感属性, 故可以直接增加蜜罐容器。

(4) 通过设定一个阈值 δ 以控制单个指纹元组的最大发布时延, 每一个数据流元组都有一个到达时间 $s.t$ 和发布时间 $s'.t$, 为了满足实时性的要求, 对于每一个元组必须满足 $s'.t-s.t<\delta$。

(5) 对不满足等价类形成条件的元组或者会引起较大信息缺失等情况出现时, 考虑从 Ψ 中重用现有的已发布等价类。

(6) 相较于传统业务, 云业务负载变化快, 容器下线可能会破坏云资源池中指纹等价类的匿名性, 因此需要对其进行检查, 对匿名性被破坏的等价类增补新容器或是蜜罐容器。

5.2 算法实现

下面介绍 CFDAA 算法的主要步骤和关键函数。算法主函数如算法 1 所示。SaaS 云网络中容器的创建、删除, 或是上线、下线行为十分频繁, 为了维持当前处于运行状态的容器满足匿名化条件, 需要考虑容器下线的情况。算法 1 首先判断哪些已发布的指纹簇的匿名化条件已经被破坏。$\psi_i(n)$ 代表第 i 个匿名簇, 其中 n 为该簇的大小, $\psi_i(n)$ 可通过 Kubernetes 等容器编排管理工具获取。算法将匿名性已破坏的容器簇重新加入候选匿名化簇的集合 Γ。类似于文献[39], 我们限定簇个数上限为 β, 牺牲一定的性能以降低复杂度(步骤 1~8)。对于当前处理元组 $s\in S_{in}$, 调用 *best_selection*() 函数计算 Γ 中与 s 距离最近的匿名化簇, 并将当前元组 s 加入该簇。如果 *best_selection*() 函数没有找到合适的匿名化簇, 则创建一个新的簇, 并将当前元组加入该簇(步骤 9~13)。最后判断是否存在达到容器滞留时间达到最大时间约束的元组 s', 若存在 s' 尚未输出, 则调用延迟控制函数 *delay_constraint*(s') 进行处理, 对其进行强制性匿名化指纹修改并输出(步骤 14~16)。

算法 1. $CFDAA(S,k,\delta,\beta,\Omega)$.

输入: 数据流 S_{in}, 匿名指标 k, 时延约束 δ, 簇个数上限 β, 当前云资源池已发布 k-匿名簇的集合 Ψ, 单容器指纹欺骗开销门限 τ;

输出: 匿名化的数据流 S_{out}

1 令 Γ 为候选匿名化簇的集合, 且初始化为 $\varnothing$

2 令 Γ_{new} 为新生成的候选匿名化簇的集合, 且初始化为 $\varnothing$

3 WHILE $S_{\text{in}}\neq\varnothing$

4 令 $s\in S_{\text{in}}$ 是当前处理的指纹元组

5 FOR EACH $\psi_i(n)\in\Psi$

6 IF $n<k$ AND $|\Gamma|<\beta$

7 将 $\psi_i(n)$ 加入 Γ_{old}, 并从 Ψ 中删除

8 $\Gamma=\Gamma_{\text{old}}+\Gamma_{\text{new}}$

9 $C_f=best\_selection(s,\beta,\tau,\Gamma)$

10 IF $C_f=NULL$ THEN

11 创建一个新簇, 将 s 放入新簇中, 并将新簇加入 Γ 中

12 ELSE

13 将 s 加入 C_f 中

14 令 s' 为到达时间为 $s'.t = s.t - \delta$ 的元组

15 IF 元组 s' 尚未输出 THEN

16 *delay_constraint*(s',k,l,Γ)

函数 *best_selection*(s,β,τ,Γ) 如算法 2 所示, 其功能是为一个指纹元组从候选匿名簇的集合 Γ 中选择最合适的簇, 元组聚类的依据是 4.2 节中定义的指纹修改开销。首先根据式(6)计算 s 加入不同候选匿名簇 C_i 后的整体指纹修改开销, 找到与 s 最接近的簇的集合 $SetC_{\min}$ (步骤 1~6)。如果 s 加入后的新簇的单个元组指纹修改开销小于阈值, 则将该簇加入候选集合 $SetC_{ok}$ (步骤 7~9)。最后, 若 $SetC_{ok}$ 不为空, 则从 $SetC_{ok}$ 随机返回一个簇大小最小的簇。若 $SetC_{ok}$ 为空, 则判断簇个数是否达到上限。若达到上限, 则从 $SetC_{\min}$ 随机返回一个簇大小最小的簇; 若未达到上限, 则返回 NULL(步骤 10~16)。

算法 2. *best_selection*(s,β,τ,Γ)

输入: 当前处理元组 s, 簇个数上限 β, 单个元组指纹修改阈值 τ, 候选匿名化簇的集合 Γ;

输出: 匿名化的数据流

1 初始化候选集合 E 为 $\varnothing$。

2 FOR RACH $C_i \in \Gamma$ DO

3 $e_i = \gamma(s \cup C_i)$

4 将 e_i 加入 E

5 $e_{\min} = \min E$

6 令 $SetC_{\min}$ 为 $e_{\min}$ 对应的簇

7 FOR RACH $C_j \in SetC_{\min}$

8 IF $\gamma(s \cup C_j) \leqslant \tau$ THEN

9 将 C_j 加入 $SetC_{ok}$

10 IF $SetC_{ok}$ 为空 THEN

11 IF $|\Gamma| \geqslant \beta$ THEN

12 从 $SetC_{\min}$ 中返回一个大小最小的簇

13 ELSE

14 RETURN NULL

15 ELSE

16 从 $SetC_{ok}$ 中含元组的总数 $\sum_{C_i \in \Gamma} |C_i|$ 进行判断, 若元组总数不超过 k, 则合并所有候选匿名簇, 再加入蜜罐容器直到簇的大小为 k, 然后调用 *supress*(C) 函数输出结果(步骤 1-3)。*supress*(C) 函数表示对容器指纹进行抑制, 即选择修改开销最小的指纹 f, 将 C 中所有容器指纹修改为 f, 然后返回结果, 并跳出 *delay_constraint*()。

若元组总数 $\sum_{C_i \in \Gamma} |C_i|$ 不小于 k, 则判断包含 s 的非 k-匿名簇 C 能够直接发布。如果簇 C 的大小不小于 k 且多样化指标 $C.diversity \geqslant l$, 则调用 *suppress*(C) 函数输出 C。其中 $C.diversity$ 代表簇 C 中不同敏感属性元组的个数, 蜜罐的敏感属性视为与正常服务不同, 且相互之间也不相同(步骤 4~7)。如果簇 C 中元组个数小于 k, 则计算不同簇之间的距离, 选择距离 C 最近簇, 并将其与 C 合并。重复执行直到簇的 C 中元组个数不小于 k(步骤 8~14)。然后判断 $C.diversity$ 是否满足多样性要求, 若不满足则加入蜜罐容器直到多样性为 l (步骤 15~17)。

算法 3. *delay_constraint*(s,k,l,Γ).

输入: 指纹元组 s, 匿名指标 k, 多样化指标 l, 候选匿名化簇的集合 Γ;

输出: 匿名化的数据流

1 IF $\sum_{C_i \in \Gamma} |C_i| \leqslant k$ THEN

2 合并 Γ 中所有簇, 再加入蜜罐容器直到簇的大小为 k, 得到簇 C_j。

3 *supress*(C_j)

4 ELSE

5 令 C 是包含 s 的候选 k-匿名簇

6 IF $|C| \geqslant k$ AND $C.diversity \geqslant l$ THEN

7 *suppress*(C)

8 ELSE

9 WHILE $|C| < k$

10 计算出距离 C 最近的簇 D。

11 IF $C \in \Gamma_{old}$ THEN

12 C 与 D 合并, 指纹修改为 C

13 IF $C \in \Gamma_{new}$ THEN

14 C 与 D 合并, 指纹修改为使得整体修改开销最小的指纹

15 WHILE $C.diversity < l$

16 C 中加入一个蜜罐容器

17 *suppress*(C)

5.3 复杂度分析

首先分析算法的时间复杂度。算法主要包括两个过程:

(1) 函数 *best_selection*() 的时间复杂度。

由于簇个数上限为 β, 则计算 s 加入不同候选匿名簇 C_i 后的整体指纹修改开销的次数小于 β。若指纹具有 $|QI|$ 个属性 $f(a_1,...,a_d)$, 最差的情况下, 计

算 s 加入不同候选匿名簇 C_i 后的整体指纹修改开销的时间复杂度为 $O(\beta\delta|QI|^2)$。判断是否满足指纹修改开销阈值和生成返回值需要的时间为 $O(1)$。则对于的数据流 S，函数 *best_selection*() 被调用 $|S|$ 次，因此函数 *best_selection*() 的时间复杂度为 $O(\beta\delta|QI|^2|S|)$。

(2) 函数 *delay_constraint*() 的时间复杂度。

实现开销主要来源于计算不同簇之间的距离，选择距离 C 最近簇，并将其与 C 合并(步骤 10)。由于设定了簇上限个数，因此时间复杂度为 $O(|QI|\beta)$。最差的情况下，*delay_constraint*() 被调用 $|S|$ 次。因此函数 *delay_constraint*() 的时间复杂度为 $O(\beta|QI||S|)$。

因此算法的总时间复杂度为 $O(\beta\delta|QI|^2|S|+\beta|QI||S|)$，由于 β 和 $|QI|$ 均远小于 $|S|$ 和 δ，则总时间复杂度可简化为 $O(\delta|S|)$。

然后分析算法的空间复杂度。

(1) 用于存储数据流的空间。若存储一个容器指纹 $f(a_1,...,a_d)$ 的空间为 $|QI|$，由于算法引入了时延控制 δ，则一般情况下用于存储数据流的空间为 $O(\delta|QI|)$。即存储新生成的候选匿名化簇的集合 Γ_{new} 的空间。

(2) 用于存储已发布但匿名性被破坏的候选匿名簇的集合 Γ_{old} 的空间。算法避免指纹二次修改，故 Γ_{old} 中每个集合只需存储一个抑制之后的指纹。由于算法对 Γ_{old} 大小进行了限制，因此空间复杂度为 $O(\beta|QI|)$。

因此算法的总空间复杂度为 $O((\beta+\delta)|QI|)$。由于 β 远小于 δ，因此总空间复杂度可简化为 $O(\delta|QI|)$。

6 实验结果及分析

为验证CFDAA算法的性能，本文采用指纹修改开销、攻击者达成攻击的开销、算法运行时间等作为评价指标，并与 CASTLE 算法[2]和 MDAV[40]的方法比较。CASTLE 算法是基于聚类的数据流匿名的代表算法，它对缓存到达的元组，采用时延控制连续发布匿名数据。MDAV 算法是一种经典的批次处理的微聚集算法，每次缓存一定数量的元组然后集中进行匿名处理。

实验数据集来源于实验室基于 Kubernetes 搭建的容器云平台的历史日志，对原始日志数据进行预处理后，得到包括操作系统，容器类型，容器大小，镜像版本，IP 段，服务端口，用户归属，应用程序类型，上线时间，下线时间等属性的数据流，敏感属性为应用程序类型。所有准标识符属性均为范畴型属性，处理后的数据流共包含 40000 多个元组。k 值设定为 100，多样性 l 设定为 8。时延控制 δ 为 2000。欺骗引擎基于 Golang 1.14 实现，方便与 Kubernetes 对接。攻击者采用 Nmap 和 X-scan 网络扫描工具对容器进行指纹探测攻击。运行容器指纹信息流式处理算法 CFDAA 的硬件配置为 Intel Core i5 3.20GHz CPU, 4Gb RAM，系统配置为 Ubuntu 12.04。

类似相关研究[1, 7]，采用攻击者成功定位目标容器概率(Attack Success Probability, ASP)的大小来度量安全性。经过匿名欺骗技术处理后容器被定位几率越小，则说明其信息安全度越高。若处理后的虚假指纹与敏感属性之间的对应关系容易被攻击者获取，则安全性低。

首先对攻防双方的能力进行假设。假定防御者根据 3.2 节中的框架设计，欺骗网关同时部署了基于 MTD 的跳变技术和指纹匿名欺骗技术；假定攻击者具有很强的网络侦查能力，获知防御者采取的防御策略，防御者不能通过指纹修改使得攻击不可达。根据文献[41]，假定采用动态轮换等机制有 e^{-1} 的概率防御攻击，则欺骗网关部署后一次攻击被成功防御的概率可表示为:

$$CM = e^{-1} + (1-e^{-1})(1-\frac{m}{h}) \tag{7}$$

其中 m 为 k-匿名簇中敏感属性的个数，h 为匿名簇的大小。若防御者仅使用指纹匿名欺骗技术，有 $CM = 1-\frac{m}{h}$。

则 n 次攻击后，目标容器被攻陷的概率为

$$T = 1-(CM)^n \tag{8}$$

假定攻击判定阈值为 T_n，则若 $T > T_n$，则认为攻击成功。可以看出攻击者的攻击开销正比于如攻击次数 $C_a \propto n$。为简化计算，令 $C_a = n$。因此攻击者实施一次成功攻击的开销为

$$C_a = \log(1-T_n)/\log CM \tag{9}$$

图 5 表示不同匿名欺骗算法对攻击达成开销的影响。三种匿名算法都能够增加攻击者的攻击成本，攻击成功的似然值越高，攻击者需要付出的代价越多。MDAV 算法没有考虑 l-多样性，导致匿名簇中可能存在多个相同类型的容器，容易遭受偏斜型攻击，

安全性最低。且 MDAV 算法是一种无记忆的批次处理算法，没有考虑之前已产生容器的安全性。CASTLE 算法可以利用已发布的高质量匿名簇降低欺骗开销，但没有考虑由于容器下线可能导致 *k*-匿名被破坏，将使得算法运行一段时间后 *CM* 的值逐步下降，进而导致实际安全性低于预期值。CFDAA 算法弥补了这一点，通过实时维护已有的容器等价类的匿名性提高系统长期运行中 *CM* 的值，虽然牺牲了一定算法性能，增大了指纹修改开销，但提高了系统安全性，增加了攻击者的攻击开销。

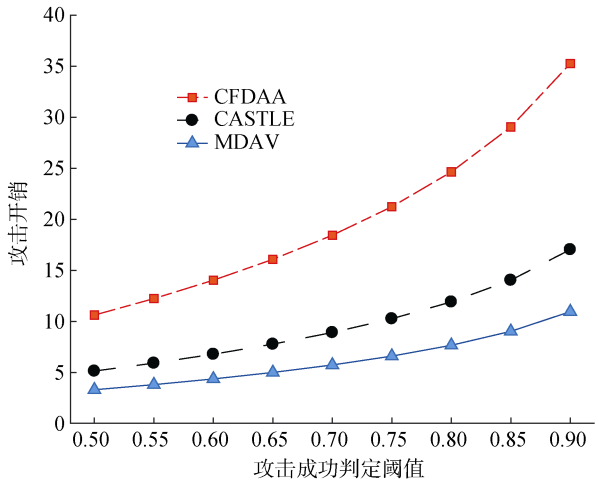

图 5　不同匿名欺骗算法下的平均攻击达成的开销
Figure 5　Average attack overheads of different anonymous algorithms

由于指纹修改需要付出一定开销，如修改网络数据流中的特征，需要对网络数据包进行修改，因此会产生一定的额外时间开销。因此如何尽可能减少对指纹的修改程度是网络欺骗需要考虑的重要问题。根据 4.2 节所定义的指纹修改开销进行度量，指纹修改开销越小，则说明指纹修改度较小，代表该技术处理后的容器指纹信息与原容器的指纹信息相似，欺骗网关对数据流的干预程度低，对系统性能影响较小。

图 6 表示不同 k 值约束下的平均指纹修改开销比较。从图中可以看出，总体而言，随着 k 值的增大，匿名簇的大小会变大，指纹修改的总体幅度上风，导致指纹修改开销不断增加。在相同的资源环境和服务请求强度下，MDAV 算法的开销最大。MDAV 无法利用已有的优质匿名组，解空间受限，使得一些容器无法获得最低开销的指纹修改方案。相比于 CASTLE 算法而言，本文提出的 CFDAA 算法具有略高的总体开销。考虑到在容器指纹匿名欺骗问题中，发布的指纹数据集并不用于统计或数据分析，故 CFDAA 并不限制匿名簇的大小。故 CFDAA 没有簇分割过程，使得其产生的匿名簇比 CASTLE 算法要大，导致指纹修改开销增大。同时，较大的匿名簇有利于减少因为容器下线而导致的匿名性被破坏。

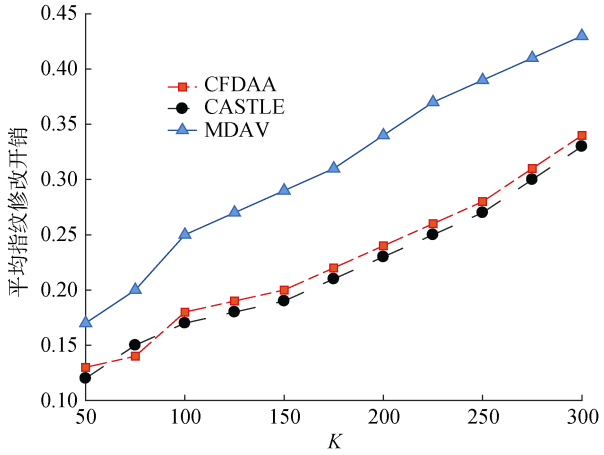

图 6　不同 *K* 值约束下的平均指纹修改开销
Figure 6　Average fingerprint modification overhead with varying *K*

图 7 表示不同时延值 δ 约束下的平均指纹修改开销。可以看出随着 δ 值增加，指纹修改开销呈下降趋势。这是因为每次修改的容器指纹越多，指纹需要大幅修改的容器占比减小，总体修改开销下降。最开始随着 δ 值的增大，平均指纹修改开销迅速下降，然后下降速度逐步减慢。由图 7 可知，所提出的 CFDAA 算法在提高指纹欺骗算法安全性的同时，没有带来过多的开销，其平均指纹修改略高于 CASTLE 算法。

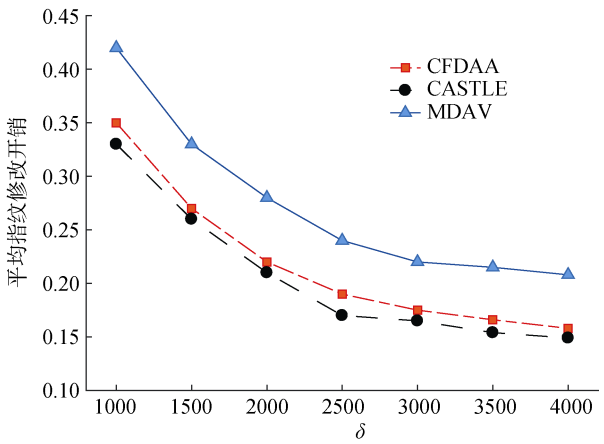

图 7　不同时延值 δ 约束下的平均指纹修改开销
Figure 7　Average fingerprint modification overhead with varying δ

不同数据量下平均指纹修改开销对比如图 8 所

示。随着需要指纹匿名的容器数量的增大, MDAV 算法的平均指纹修改开销基本稳定, 这是因为 MDAV 是一种无记忆的批次处理算法, 无法利用已经存在优质匿名簇, 算法输出结果与要处理的数据流大小无关。CASTLE 算法和 CFDAA 算法随着数据量增长, 积累的优质簇数量逐渐增多, 因此平均指纹修改开销都随着数据量逐渐降低。CASTLE 算法的下降幅度略大于 CFDAA 算法, 这是由于 CASTLE 算法建立了一种优质匿名簇复用机制, 只挑选指纹修改开销小的簇进行复用, 而 CFDAA 算法主要复用因容器下线而导致匿名性被破坏的簇。为直观显示所提方案的性能, $T_n=0.7$, $\delta=2000$, 数据量为 20000 时不同欺骗算法的攻击开销和平均指纹修改开销如图 9 所示。可以看出相较于对比算法, CFDAA 算法在略微增加指纹修改开销的情况下显著增大了攻击者的攻击开销, 具有较高的部署性价比。

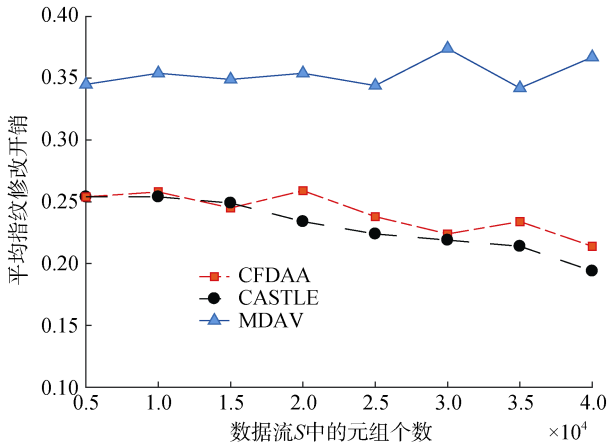

图 8　不同数据量下平均指纹修改开销对比

Figure 8　Average fingerprint modification overhead with varying S

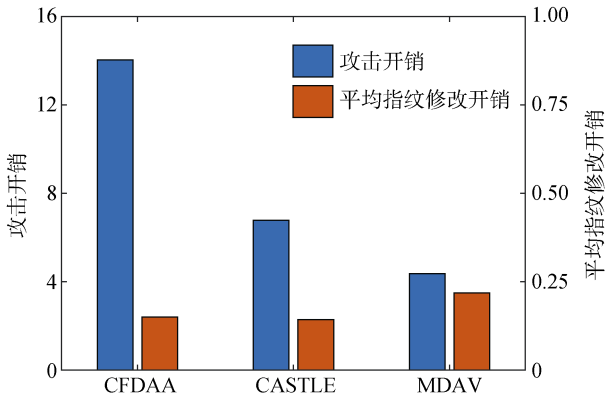

图 9　T_n= 0.7 时不同匿名欺骗算法的攻击开销和平均指纹修改开销

Figure 9　Attack overhead and fingerprint modification overhead of different algorithms when T_n= 0.7

不同 δ 值约束下算法运行时间情况如图 10 所示。算法的运行时间随着 δ 值的增大而上升, 由 5.3 节的复杂度分析可知, CFDAA 时间复杂度与 δ 值正相关。可以看出 MDAV 算法运行速度最快, 因为其逻辑简单, 没有复杂的簇合并和分割操作, 也不用维护已发布的匿名数据集。CFDAA 算法的时间开销小于 CASTLE 算法, 这是因为 CFDAA 摒弃了 CASTLE 中时间开销较大的对较大匿名簇的分割操作。CASTLE 算法设计初衷是应对数据集脱敏发布问题, 即对数据进行匿名的同时保持数据的统计意义, 然后公开发布匿名后数据集供其他机构使用。而在指纹匿名问题中, 发布数据不需要保持原有统计意义, 无需降低簇的大小。

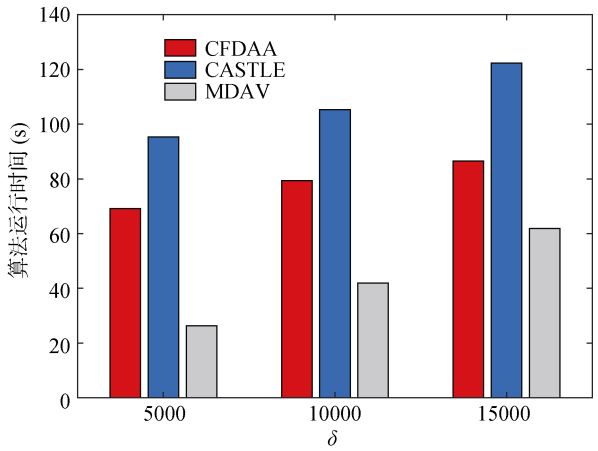

图 10　不同δ值约束下运行时间对比图

Figure 10　Running time with varying δ

7　结论

云网络安全事件频发逐渐引起学术界和工业界的广泛关注, 针对目标云资源的定位是网络攻击的前置步骤。作为一种主动防御技术, 网络欺骗可以一定程度上扭转攻防双方的不对称态势。然而, 现有的网络欺骗方法存在开销巨大、安全性不足、难以应对高级攻击者等问题。本文提出了一种基于容器指纹匿名的网络欺骗方法, 该算法利用多种欺骗技术的组合来对抗网络侦察。具体来说, 该算法可以对云环境中连续在线的容器执行实时匿名处理。鉴于需要修改伪装的容器指纹信息是一个持续产生的数据流, 提出一种基于时延约束和簇分割的容器指纹信息流式处理算法 CFDAA, 在满足实时性约束的条件下, 输出开销最小的容器指纹修改方案。实验结果表明所提出方法能够对 SaaS 云环境中动态上线的容器进行实时匿名, 并保证云资源池中的容器始终满足

匿名约束。所提匿名在线算法在安全性能上优于传统方法, 且修改开销可以接受。

本文的最后, 我们提出了一些该领域值得进一步挖掘的研究点:

(1) 本文仅考虑对容器服务类型进行隐藏, 即敏感属性的个数为 1, 而对于设备的隐私属性经常不止一个, 用户身份, 安全等级要求等都可能作为隐私属性。在未来的工作中, 可以扩展对容器的多种隐藏属性进行匿名隐藏。

(2) 本文将其他业务的容器作为关键容器的影子容器, 所提算法没有考虑这些容器被攻击导致的服务终端、信息泄露等方面的损失。本文仅从增加攻击者的攻击开销方面分析了算法的安全性能。鉴于蜜罐可以捕获和分析攻击行为, 因此当攻击者攻击其目标容器、其他容器和蜜罐容器这三种情况时, 防御者具有不同的安全损失或收益。从而可以基于防御收益对容器指纹欺骗算法进行优化和评估。

(3) 网络安全已成为物联网与各行业的融合发展的主要阻碍之一。现有研究主要集中于基于设备指纹进行身份验证和接入控制, 鉴于物联网蜜罐的出现[42], 本文提出的基于匿名的欺骗策略或可应用于物联网场景, 降低物联网设备被攻击者定位和追踪的风险。

参考文献

[1] Hutchins E, Cloppert M J, Amin R M. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains [J]. *Leading Issues in Information Warfare & Security Research*, 2011, 1(1): 80.

[2] Cao J N, Carminati B, Ferrari E, et al. CASTLE: Continuously Anonymizing Data Streams[J]. *IEEE Transactions on Dependable and Secure Computing*, 2011, 8(3): 337-352.

[3] Yousefpour A, Fung C, Nguyen T, et al. All one Needs to Know about Fog Computing and Related Edge Computing Paradigms: A Complete Survey[J]. *Journal of Systems Architecture*, 2019, 98: 289-330.

[4] Jia Z P, Fang B X, Liu C G, et al. Survey on Cyber Deception[J]. *Journal on Communications*, 2017, 38(12): 128-143.
(贾召鹏, 方滨兴, 刘潮歌, 等. 网络欺骗技术综述[J]. *通信学报*, 2017, 38(12): 128-143.)

[5] Feng X, Li Q, Wang H, et al. Acquisitional rule-based engine for discovering internet-of-thing devices[C]. *27th USENIX Security Symposium*, 2018: 327-341.

[6] Gaël Roualland & Jean-Marc Saffroy. IP Personality, http://ippersonality.sourceforge.net, November, 2020.

[7] Sharma D P, Kim D S, Yoon S, et al. FRVM: Flexible Random Virtual IP Multiplexing in Software-Defined Networks[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*, 2018: 579-587.

[8] Shu Z, Yan G H. Ensuring Deception Consistency for FTP Services Hardened Against Advanced Persistent Threats[C]. *The 5th ACM Workshop on Moving Target Defense*, 2018: 69-79.

[9] Jafarian J H H, Al-Shaer E, Duan Q. Spatio-Temporal Address Mutation for Proactive Cyber Agility Against Sophisticated Attackers[C]. *The First ACM Workshop on Moving Target Defense*, 2014: 69-78.

[10] You J Z, Lv S C, Sun Y Y, et al. A Survey on Honeypots of Internet of Things[J]. *Journal of Cyber Security*, 2020, 5(4): 138-156.
(游建舟, 吕世超, 孙玉砚, 等. 物联网蜜罐综述[J]. *信息安全学报*, 2020, 5(4): 138-156.)

[11] Jafarian J H, Niakanlahiji A, Al-Shaer E, et al. Multi-Dimensional Host Identity Anonymization for Defeating Skilled Attackers[C]. *The 2016 ACM Workshop on Moving Target Defense*, 2016: 47-58.

[12] Duan Q, Al-Shaer E, Islam M. CONCEAL: A Strategy Composition for Resilient Cyber Deception: Framework, Metrics, and Deployment*Autonomous Cyber Deception*, 2018: 1-9.

[13] Verma A, Kaushal S. Cloud Computing Security Issues and Challenges: A Survey[C]. *Advances in Computing and Communications*, 2011: 445-454.

[14] Stoll C. The Cuckoo's Egg: Tracking A Spy Through The Maze Of Computer Espionage[M]. Simon and Schuster, 2005.

[15] Kreibich C, Crowcroft J. Honeycomb[J]. *ACM SIGCOMM Computer Communication Review*, 2004, 34(1): 51-56.

[16] Hu Y J, Ma J, Guo Y B, et al. Research on Active Defense Based on Multi-Stage Cyber Deception Game[J]. *Journal on Communications*, 2020, 41(8): 32-42.
(胡永进, 马骏, 郭渊博, 等. 基于多阶段网络欺骗博弈的主动防御研究[J]. *通信学报*, 2020, 41(8): 32-42.)

[17] Fraunholz, D., Anton, S. D., Lipps, C., Reti, D., Krohmer, D., Pohl, F., & Schotten, H. D. Demystifying deception technology: A survey[EB/OL]. 2018: arXiv preprint arXiv:1804.06196.

[18] Wang P C, Chen F C, Cheng G Z, et al. L2/L3 Address Cooperative Mimicry Strategy Research Based on SDN[J]. *Acta Electronica Sinica*, 2019, 47(10): 2032-2039.
(王鹏超, 陈福才, 程国振, 等. 软件定义的 L2/L3 地址协同拟态伪装策略研究[J]. *电子学报*, 2019, 47(10): 2032-2039.)

[19] Albanese M, Battista E, Jajodia S. A Deception Based Approach for Defeating OS and Service Fingerprinting[C]. *2015 IEEE Conference on Communications and Network Security*, 2015: 317-325.

[20] Cho J H, Sharma D P, Alavizadeh H, et al. Toward Proactive, Adaptive Defense: A Survey on Moving Target Defense[J]. *IEEE Communications Surveys & Tutorials*, 2020, 22(1): 709-745.

[21] Anderson B, McGrew D. OS Fingerprinting: New Techniques and a Study of Information Gain and Obfuscation[C]. *2017 IEEE Conference on Communications and Network Security*, 2017: 1-9.

[22] Laperdrix P, Rudametkin W, Baudry B. Mitigating Browser Fingerprint Tracking: Multi-Level Reconfiguration and Diversification[C]. *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2015: 98-108.

[23] Park K, Woo S, Moon D, et al. Secure Cyber Deception Architecture and Decoy Injection to Mitigate the Insider Threat[J]. *Symme-*

try, 2018, 10(1): 14.

[24] Zhao Z, Liu F L, Gong D F. An SDN-Based Fingerprint Hopping Method to Prevent Fingerprinting Attacks[J]. *Security and Communication Networks*, 2017(2):1-12.

[25] Fusys & CyRaX, Fingerprint Fucker, http://www.s0ftpj.org/tools/fingfuck.tgz, November, 2020.

[26] Rahman M A, Manshaei M H, Al-Shaer E. A Game-Theoretic Approach for Deceiving Remote Operating System Fingerprinting[C]. *2013 IEEE Conference on Communications and Network Security*, 2013: 73-81.

[27] Vetterl A, Clayton R. Honware: A Virtual Honeypot Framework for Capturing CPE and IoT Zero Days[C]. *2019 APWG Symposium on Electronic Crime Research*, 2019: 1-13.

[28] Liu C G, Fang B X, Liu B X, et al. A Hierarchical Model of Targeted Cyber Attacks Attribution[J]. *Journal of Cyber Security*, 2019, 4(4): 1-18.
(刘潮歌, 方滨兴, 刘宝旭, 等. 定向网络攻击追踪溯源层次化模型研究[J]. *信息安全学报*, 2019, 4(4): 1-18.)

[29] Leita C, Mermoud K, Dacier M. ScriptGen: An Automated Script Generation Tool for Honeyd[C]. *21st Annual Computer Security Applications Conference*, 2005: 12-24.

[30] Fan W J, Fernández D, Du Z H. Adaptive and Flexible Virtual Honeynet[C]. *MSPN 2015*: *Selected Papers of the First International Conference on Mobile*, *Secure*, *and Programmable Networking - Volume 9395*, 2015: 1-17.

[31] Fan W J, Fernández D, Du Z H. Versatile Virtual Honeynet Management Framework[J]. *IET Information Security*, 2017, 11(1): 38-45.

[32] Fraunholz D, Zimmermann M, Schotten H D. An Adaptive Honeypot Configuration, Deployment and Maintenance Strategy[C]. *2017 19th International Conference on Advanced Communication Technology*, 2017: 53-57.

[33] Sun J H, Sun K. DESIR: Decoy-Enhanced Seamless IP Randomization[C]. *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016: 1-9.

[34] Zhang L C, Wang Z X, Fang J B, et al. A SDN Proactive Defense Scheme Based on IP and MAC Address Mutation[C]. *Wireless Internet*, 2016: 51-60.

[35] Yang K C, Liu J Q, Zhang C, et al. Adversarial Examples Against the Deep Learning Based Network Intrusion Detection Systems[C]. *MILCOM 2018 - 2018 IEEE Military Communications Conference*, 2018: 559-564.

[36] Hu Y J, Guo Y B, Ma J, et al. Method to Generate Cyber Deception Traffic Based on Adversarial Sample[J]. *Journal on Communications*, 2020, 41(9): 59-70.
(胡永进, 郭渊博, 马骏, 等. 基于对抗样本的网络欺骗流量生成方法[J]. *通信学报*, 2020, 41(9): 59-70.)

[37] Samarati P, Sweeney L. Protecting Privacy when Disclosing Information: K-Anonymity and Its Enforcement through Generalization and Suppression. Technical Report, SRI Computer Science Lab., 1998.

[38] Guo K, Zhang Q S. Fast Clustering-Based Anonymization Algorithm for Data Streams[J]. *Journal of Software*, 2013, 24(8): 1852-1867.
(郭昆, 张岐山. 基于聚类的快速数据流匿名方法[J]. *软件学报*, 2013, 24(8): 1852-1867.)

[39] Wang P, Lu J J, Zhao L, et al. B-CASTLE: An Efficient Publishing Algorithm for K-Anonymizing Data Streams[C]. *2010 Second WRI Global Congress on Intelligent Systems*, 2010: 132-136.

[40] Solanas A, González-Nicolás Ú, Martínez-Ballesté A. A Variable-MDAV-Based Partitioning Strategy to Continuous Multivariate Microaggregation with Genetic Algorithms[C]. *The 2010 International Joint Conference on Neural Networks*, 2010: 1-7.

[41] Jafarian J H, Al-Shaer E, Duan Q. Openflow Random Host Mutation: Transparent Moving Target Defense Using Software Defined Networking[C]. *The first workshop on Hot topics in software defined networks*, 2012: 127-132.

[42] Pa Y M P, Suzuki S, Yoshioka K, et al. IoTPOT: Analysing the Rise of IoT Compromises[C]. *Usenix Conference on Offensive Technologies. USENIX Association*, 2015:1-9.

李凌书 2017 年在解放军信息工程大学信息与通信工程专业获得硕士学位。现在解放军信息工程大学学校网络空间安全专业攻读博士学位。研究领域为网络主动防御、网络欺骗。Email: lls.ndsc@aliyun.com

刘文彦 解放军信息工程大学信息技术研究所讲师, 博士。研究领域为云计算、网络功能虚拟化。研究兴趣包括: 云资源管理与调度。Email: lwynudt@163.com

邬江兴 解放军信息工程大学信息技术研究所主任, 教授。研究领域为拟态安全防御、内生安全防御。Email: jiangxing.wu@outlook.com