软件定义网络中资源消耗型攻击及防御综述

徐建峰[1,2], 王利明[1], 徐 震[1]

[1] 中国科学院信息工程研究所, 北京 中国 100093
[2] 中国科学院大学 网络空间安全学院, 北京 中国 100049

摘要 在传统网络中, 集成多种网络功能的控制平面和负责转发数据包的数据平面是紧密耦合的, 并且通常嵌入在一个专用设备中, 这严重限制了网络管理的灵活性和网络服务的创新性。软件定义网络(Software-Defined Networking, SDN)作为一种新型的网络范式, 通过将控制平面与数据平面解耦克服了传统网络架构的不足。研究人员凭借全网视图可见性以及对网络设备直接编程的能力提出了诸多 SDN 应用场景, 如数据中心网络、云和广域网。然而 SDN 带来的灵活性、可管理性以及可编程性等优点是以引入新的安全挑战为代价。本文聚焦 SDN 网络中的资源消耗型攻击, 首先分层整理了 SDN 网络中的关键资源以及攻击目标, 然后对控制平面、控制通道和数据平面存在的多种资源消耗型攻击以及现有防御机制做出了详细的分析和归纳, 最后对未来的研究工作进行了展望, 并提出了一些潜在的研究方向。

关键词 软件定义网络; 安全; 资源消耗型攻击
中图法分类号 TP393 **DOI 号** 10.19363/J.cnki.cn10-1380/tn.2020.07.06

Survey on Resource Consumption Attacks and Defenses in Software-Defined Networking

XU Jianfeng[1,2], WANG Liming[1], XU Zhen[1]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
[2]School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

Abstract In traditional networks, a control plane integrating various network functions and a data plane responsible for forwarding packets, are tightly coupled and typically embedded within a single proprietary device, which severely limits the flexibility of network management and the potential for network service innovation. Software-Defined Networking (SDN), as a promising network paradigm, circumvents these deficiencies in traditional networks via decoupling the control plane from the data plane. With the excitement of holistic visibility across the network and the ability to program network devices directly, researchers have rushed to present a range of new SDN-enhanced application scenarios, such as data center networks, cloud and wide area networks. The flexibility, manageability and programmability brought by SDN, however, come at the cost of new security challenges. This paper focuses on resource consumption attacks in SDN networks. It first introduces the key resources and attack targets in SDN, and then summarizes and analyzes the possible consumption attacks and existing countermeasures in control plane, control channel, and data plane. Finally, this survey paper discusses some future works and suggested research directions.

Key words software-defined networking (SDN); security; resource consumption attack

1 引言

传统 IP 网络运行在多种分布式控制协议之上, 庞大的分布式协议体系使得网络变得相当复杂和难以管理。与此同时, 随着网络业务的创新和发展, 诸多新型服务如网络虚拟化, 服务质量(Quality of Service, QoS), 软件即服务(Software-as-a-Service, SaaS)等不断涌现出来, 它们需要网络运维人员根据用户需求频繁地更改设备配置以及编排网络资源。然而, 僵化的传统网络架构难以适应上述动态的用户需求, 这严重阻碍了网络服务的演进和创新。例如 IPv4 向 IPv6 的过渡, 该工作始于十多年前, 如今仍然没有彻底完成, 而两者之间的过渡本质上只是路由协议的更新。究其原因, 传统网络设备中的数据平面(负责报文转发)与控制平面(控制逻辑)是紧密耦合的, 这种大型主机式的设备架构严重限制了未来网络的

通讯作者: 王利明, 博士, 正高级工程师, Email: wangliming@iie.ac.cn。
本课题得到中国科学院战略性先导科技专项(No.Y9W0011505)资助。
收稿日期: 2019-12-20; 修改日期: 2020-02-09; 定稿日期: 2020-06-19

发展前景[1]。

2006 年, 美国斯坦福大学联合国家自然科学基金会(National Science Foundation, NSF)启动了 Clean Slate 项目, 其目的是探索出一个新型网络架构, 以摆脱目前网络面临的困境。2008 年, 斯坦福大学 The McKeown Group 团队在项目中提出了软件定义网络[2](Software-Defined Networking, SDN)的概念。SDN 作为一种新型的网络创新架构, 有望改变当前网络基础设施的局限性。首先, 它将网络控制逻辑从底层设备中分离出去, 打破了网络设备的垂直耦合。其次, 通过控制平面和数据平面的分离, 控制逻辑在集中式控制器(或网络操作系统)中实现, 从而简化了策略实施和网络配置/重配置。一经提出, SDN 便迅速获得了学术界和工业界的广泛关注, 诸多 SDN 应用场景被研究人员提出, 如企业网[3]、数据中心[4]、云[5]和 IoT[6]等。例如 Google 部署了一个互连其全球数据中心的 SDN 网络, 称为 B4[7], 该网络极大地提高了运营效率并显著降低了运维成本; VMware 的网络虚拟化平台 NSX[8]基于 SDN 原则为用户提供了一个独立于底层网络设备的虚拟网络。近年来, Microsoft SWAN[9]、Virtual Networks[10]以及 SDN-WAN[11]的相继提出彰显了 SDN 与广域网相融合的趋势与机遇。

随着 SDN 技术的发展, 传统网络设备很可能会被 SDN 应用程序和白盒设备所取代[12]。然而传统网络大规模迁移的障碍之一是网络运维人员对 SDN 架构安全性的担忧。一方面, SDN 架构为网络基础设施提供了良好的可管理性、灵活性和可编程性, 使得运维人员能够高效地执行网络监控、网络资源分配以及动态网络重配置。另一方面, SDN 架构为网络引入了诸多新型的安全威胁, 使得攻击者能够轻易地操纵或滥用控制逻辑, 进而破坏正常的网络功能。此外, 由于集中控制的特点, 面向 SDN 架构的攻击可能会对网络造成更为严重的破坏。例如 SDN 控制器作为 ARP 代理维护着一个全局 ARP 缓存, 当发生 ARP 欺骗攻击时, 全局缓存受到污染的影响远比传统网络中单一设备遭到 ARP 攻击的影响广泛。

随着 SDN 安全问题的不断曝出, 各企业厂商、研究团体及标准化组织纷纷启动了相关方面的研究, 其中包括恶意网络应用检测、应用程序与控制器之间的认证与访问控制机制、SDN 网络机密信息保护、流规则冲突避免以及控制器或交换机的高可用保护等[1]。不同于其他文献, 本文聚焦于 SDN 中的资源消耗型攻击, 它是拒绝服务攻击的一种典型实现形式。攻击者通过消耗 SDN 网络中的关键资源来降低网络服务质量, 甚至破坏服务的可用性。例如恶意应用程序过多占用控制器计算资源, 从而导致控制器对交换机请求消息的处理效率急剧下降。

本文参考攻击实施的一般流程, 分别论述了资源消耗型攻击中的攻击目标识别过程(即关键资源识别)、攻击实施过程以及攻击效果产生过程。同时本文依据分层思想, 归纳整理了现有安全机制的防御思路并分析了它们的不足之处。文章结构安排如下: 第二章介绍 SDN/OpenFlow 的背景知识以及 SDN 安全研究现状; 第三章对 SDN 网络中的潜在攻击目标进行总结并根据 CIA(Confidentiality, Integrity and Availability)模型获得对应的安全需求; 第四章综述现有文献中提出的资源消耗型攻击, 并通过 Open vSwitch[13] 和 Ryu 控制器[14]搭建的实验平台验证攻击的可行性与有效性; 第五章对资源消耗型攻击的现有防御技术进行分类论述; 第六章对 SDN 网络中的资源消耗型攻击和防御机制进行总结和讨论。从攻击角度, 描述了资源消耗型攻击的演进方向。从防御角度, 分析了现有防御技术的不足以及未来优化和改进的思路; 第七章对全文进行总结。

2 研究背景

本节回顾了 SDN 网络体系架构和 OpenFlow 协议[2]。现实中存在一种误解, 即 SDN 和 OpenFlow 是等价的。事实上, SDN 是一种新兴的网络架构, 其中控制平面与数据平面相分离, 并且实现了对网络设备的直接编程, 而 OpenFlow 本质上是控制平面和数据平面之间定义的一个南向通信协议。此外, 本节简要描述了 SDN 的安全研究现状。

2.1 软件定义网络

传统网络设备(如交换机)中包含两个关键组件: 控制平面和数据平面。控制平面负责执行创建路由表等复杂的网络功能以确定网络流量的控制策略。数据平面则根据来自控制平面的策略完成网络数据包的处理任务。由于两个平面的紧密耦合, 当网络运维人员希望添加新的网络功能或协议时, 必须先删除旧设备, 然后实施和再次部署重新设计过的新设备, 这严重阻碍了新的或先进的网络功能和协议的快速测试和部署。

SDN 作为一种新兴的可编程网络架构, 试图通过分离控制平面和数据平面来解决上述问题。因此 SDN 简化了网络设备的设计, 使其仅提供基础且高效的数据包转发功能。复杂、动态、多样的控制平面协议或功能被整合为软件应用程序, 托管在逻辑集中的控制平台(即 SDN 控制器)上。控制平面和数

据平面分离的体系架构使得网络运维人员能够通过操纵应用程序的方式便捷灵活地部署、升级、移除网络功能与服务。

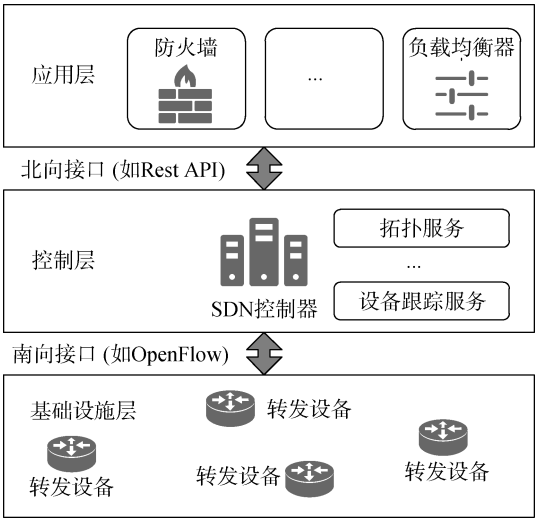


图 1 SDN 网络架构图
Figure 1 SDN architecture

按照开放网络基金(Open Networking Foundation, ONF)的定义, SDN 网络架构分为三层, 如图 1 所示。基础设施层表示数据平面, 由网络底层的转发设备组成, 主要负责数据包的处理、转发以及流量统计信息采集。控制层作为 SDN 的大脑, 一方面利用南向接口获取底层基础设施信息, 维护全网拓扑以及网络状态信息, 另一方面为开发人员提供开发网络应用程序的北向接口, 控制数据包转发。网络应用程序被部署在应用层, 实现多样化的网络功能, 如路由、网络监控、异常检测和负载平衡。控制层和应用层统称为 SDN 控制平面。

2.2 SDN 南向协议

在 SDN 中, 由于控制平面和数据平面的解耦, 两者之间的通信方式由单个设备中的内部通信转变为两个独立设备之间的远程通信。因此许多优秀的通信协议被提出, 其中 OpenFlow 作为事实上的南向通信标准已被成功应用于诸多商业部署中[7]。在 OpenFlow 网络中, 逻辑集中的控制平面(即控制器)与多个 OpenFlow 交换机建立安全连接, 并且通过该连接下发流规则, 从而动态地配置交换机的转发行为。如图 2 所示, 流规则包含了用来识别数据包的匹配字段, 一组描述数据包处理动作的指令字段, 以及统计数据包个数和字节数的计数器字段等。当交换机接收到数据包时, 首先查询自身流表是否拥有与该数据包匹配的流规则。如果存在这样的流规则, 交换机会按照拥有最高优先级的匹配规则处理此数据包, 否则交换机利用 Packet-In 事件将数据包引导至控制器, 由控制器上运行的网络应用程序对该数据包进行处理并产生转发决策。控制器使用 Flow-Mod 消息向交换机下发相应的转发决策, 交换机根据该决策对当前数据包以及后续数据包进行处理(如转发至特定端口)。这种响应式数据包处理在 OpenFlow 网络中得到了广泛的应用[15]。

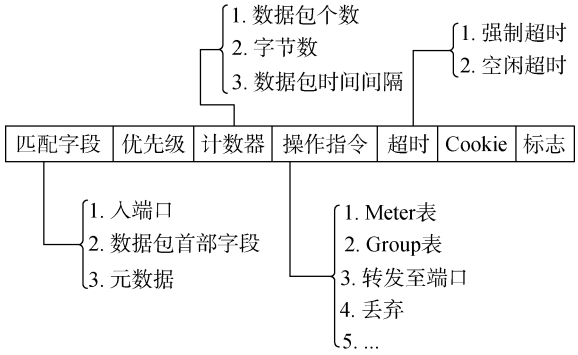


图 2 OpenFlow 协议中的标准流表项
Figure 2 Standard flow entries in the OpenFlow protocol

此外, SDN 经过十多年的深入发展, 南向协议由标准化组织以及主流设备厂家不断完善演进, 已很好地实现了与网络设备的对接, 为 SDN 的发展提供坚实基础。NETCONF(Network Configuration Protocol)[16]由 RFC 6241 进行定义和规范, 为网络设备安装、操作、配置信息以及删除等提供相关工作机制。在 SDN 中, 控制器通过使用 NETCONF 可以方便灵活地对网络设备进行管理和配置, 为传统网络像 SDN 网络迁移提供便利。PCEP (Path Computation Element Communication Protocol)[17]路径计算单元协议是由 IETF PCE 工作组提出的一种支持集中化路径计算的路径信息传输协议, 其目的是为了解决 MPLS 流量工程无法自动计算路径的限制。通过扩展支持 SR 技术, SDN 控制器计算完转发路径后, 实现 MPLS 标签转换成 SR 标签, 并利用 PCEP 协议下发给网络设备, 设备通过 SR-TE 转发路径进行数据包的转发。OVSDB 管理协议(Open vSwitch Database Management Protocol)[18]最初由 VMware 提出, 作为实现对虚拟交换机可编程访问和配置管理的 SDN 南向协议。BGP-LS (BGP Link-State)[19]由 RFC 7752 定义的, 是 MP-BGP 的进一步扩展, 在 SDN 网络中主要用于网络拓扑收集, 使得拓扑收集更加简单高效。XMPP(eXtensible Messaging and Presence Protocol)[20]主要用于即时消息传递、状态、多方聊天、语音和视频呼叫等功能, OpenContrail 控制器利用该协议与

vRouter 进行信息交互。由此可以看出, 主流的南向接口由于技术特点不同, 其应用的网络场景也不尽相同。OpenFlow 作为 SDN 网络中的代表性南向协议, 大部分现有文献是以研究 OpenFlow 协议为主, 因此本文所关注的资源消耗型攻击同样是针对采用 OpenFlow 南向协议的 SDN 网络。

表 1 SDN 中的潜在攻击目标

Table 1 Potential attacked targets in SDN

层次	潜在威胁	目标资源
控制平面	机密性	控制器配置、网络服务配置、全局网络拓扑等敏感网络状态信息
	完整性	控制器配置、网络服务配置、全局网络拓扑, 活动网络服务列表等网络状态信息
	可用性	控制器实例、控制器基本服务实例、网络应用实例
控制通道	机密性	控制消息
	完整性	控制消息
	可用性	控制通道连接
数据平面	机密性	流规则
	完整性	流规则
	可用性	交换机实例

2.3 SDN 安全研究

随着 SDN 网络不断发展, SDN 安全逐渐成为学术界和工业界研究的热点问题, 现有的 SDN 安全研究可分为以下两个方向。

第一, 利用集中式控制平面实现安全逻辑来解决传统网络中的问题。现有研究包括网络监控和测量[21-23]、访问控制[24]、防火墙[25-26]、入侵检测及防御系统[27]、DDOS 检测[28-29]、组合式安全服务[30]等。

第二, 发现和解决 SDN 架构本身的安全问题。Avant-Guard[31], FloodGuard[32], FloodDefender[33]以及 LineSwitch[34]缓解了控制平面的拒绝服务攻击。FortNOX[35]和 SE-Floodlight[36]设计了控制器的安全实施内核。Rosemary[37]使用沙箱来防止恶意应用程序影响整个控制平面, 从而增强控制平面的可靠性。ONIX[38]和 Ravana[39]采用分布式控制器结构提高了控制平面的可扩展性。DELTA[40]是一个基于模糊测试的渗透框架, 用来发现 SDN 控制器的安全问题。ConGuard[41]检测了 SDN 控制器中的有害资源竞争条件。FlowChecker[42], FLOVER[43]以及 PGA[44]发现并解决流规则冲突问题。ProvSDN[15]检测 SDN 应用层中的非特权应用程序通过操纵共享控制平面状态欺骗特权应用程序发起攻击的“借刀杀人”行为。TopoGuard[45]和 TopoGuard+[46]预防了网络拓扑中毒攻击。SPHINX[47], WedgeTail[48]以及 DYNAPFV[49]用来检测 SDN 数据平面中的恶意交换机。ATTAIN[50]是一种基于 OpenFlow 的攻击注入框架, 用来检测 SDN 架构安全性。BEADS[51]通过自动生成测试场景检测不遵守 OpenFlow 协议的恶意交换机和主机等。

本文归属于 SDN 安全研究的第二个方向, 即研究 SDN 架构本身存在的安全问题。文章综述了 SDN 网络中一种典型的拒绝服务攻击, 即资源消耗类攻击。攻击者通过多种方式过度占用 SDN 网络中的关键资源, 从而使得网络服务质量急剧下降, 甚至完全丧失, 如通过拥塞控制器和交换机之间的控制通道来增加合法主机之间的通信时延。

3 潜在攻击目标

本节介绍了 SDN 网络架构中控制平面、控制通道以及数据平面的潜在攻击目标, 并应用 CIA 三元组模型获得对应的安全需求, 相关信息总结于表 1。

3.1 控制平面

控制平面的主要作用是响应来自数据平面的数据包处理请求, 因此控制平面一旦崩溃, 整个网络就会失去控制。控制平面的关键资源之一是控制器实例本身。当前 SDN 控制器实例通常是运行在终端主机上的一个用户级应用程序, 它应该始终处于可用状态。此外, 控制器配置信息的泄露和非法篡改也会为网络带来风险。控制器配置信息通常包含非常敏感的信息, 如管理员登录凭证和 REST API 访问 URL 等, 攻击者可以使用这些敏感信息来接管整个网络。控制器本身还会提供一些基础网络服务, 如设备管理服务、主机追踪服务以及拓扑管理服务等, 运维人员应始终保证这些基础服务的可用性。

控制平面中的另一个关键资源是网络服务实例(即 SDN 应用程序), 它们实现了重要的网络功能, 包括简单的转发以及高级的负载平衡。这些服务应始终保证可用, 否则正常的网络行为将会受到影响。

此外, 网络服务也是可配置的, 它们的配置信息同样不应该被公开或恶意操纵。例如披露或操作包含防火墙应用 ACL(Access Control List)规则的配置信息会使整个网络处于危险之中, 以及随意更改 ARP 缓存信息会导致欺骗攻击的发生[52]。

控制平面中还维护着一些重要的网络状态信息, 包括全局网络视图、活动网络服务列表、主机位置信息以及设备配置信息等。SDN 应用程序通常会使用这些网络状态信息做出控制决策, 如响应式路由

服务会根据全局网络拓扑信息来计算通信主机之间的最短路径, 因此中毒的网络视图会误导路由服务的正常运行[45]。同时运维人员应保证网络拓扑等敏感信息的机密性, 确保不被攻击者窃取。

3.2 控制通道

SDN 控制器与多个网络设备建立连接并通过该连接交换控制消息, 如流规则安装消息。控制消息中通常包含敏感的网络信息和重要的控制决策。攻击者可以通过窃听或者篡改控制消息来获得重要的网络信息甚至破坏网络功能, 因此运维人员必须保证这些消息的机密性和完整性。此外, 控制器与网络设备之间的控制通道连接也可能会因为遭受攻击而断开, 运维人员应始终保证该连接的可用性。

3.3 数据平面

数据平面组件(SDN 交换机)与集中式 SDN 控制器建立连接, 并通过交换控制消息与控制器通信。SDN 交换机应始终能够正常地接受/发送控制消息并按照控制器的指示安装流规则, 因此在任何情况下 SDN 交换机都应处于可用状态。但是由于具有较差安全缺省值的操作系统、过时的软件以及缺少补丁更新[51], SDN 交换机很容易被攻击者入侵。

控制器通过将流规则安装到 SDN 交换机来实现网络编程。攻击者可以对安装到交换机中的流规则进行未授权篡改, 进而更改网络转发行为甚至完全破坏网络功能。此外, 因为流规则直接描述了整个网络的控制策略, 所以这些规则的机密性也必须得到保证, 否则攻击者通过窃取流规则可以轻易地推测出防火墙 ACL 策略或路由等机密信息[53-55]。

4 资源消耗攻击

本节综述了控制平面、控制通道和数据平面中存在的多种资源消耗攻击, 如表 2 所示。针对每一种攻击, 描述了攻击者模型以及攻击原理, 并在基于 Ryu 控制器和 Open vSwitch 搭建的实验平台上验证了攻击的可行性和有效性。

4.1 控制平面攻击

当前 SDN 控制平面本质上是终端主机上的一个用户级应用程序, 它的性能受到了主机资源的限制。攻击者可以通过消耗宿主机的计算资源和内存资源来破坏控制器实例的可用性。

4.1.1 Packet-In 泛洪攻击

攻击者能力. 攻击者无需利用控制器或者交换机的漏洞来入侵设备, 南向接口和北向接口同时得到了安全协议的保护, 如 HTTPS 和 SSL\TLS 协议, 攻击者仅采用恶意软件等方式入侵一个或多个 SDN 网络中的主机, 并能够控制入侵主机通过地址伪造来产生虚假数据包, 从而触发 SDN 交换机向控制器发送大量的 Packet-In 消息。

表 2 SDN 中的资源消耗型攻击

Table 2 Resource consumption attacks in SDN

攻击目标	资源类型	攻击类型	攻击简介
控制平面	计算资源	Packet-In 泛洪攻击	攻击者通过随机伪造数据包, 短时间内触发大规模的 Packet-In 消息进入控制器, 导致控制器消耗大量计算资源来处理这些无效的 Packet-In 消息。
		Table-miss 增强攻击	一种隐蔽性好, 攻击效率高的特殊 Packet-In 泛洪攻击。
		恶意应用资源耗尽攻击	包含恶意控制逻辑的 SDN 应用程序消耗控制器资源。
	内存资源	交换机表溢出攻击	通过伪造交换机的身份信息, 使得控制器中保存交换机信息的数据结构溢出, 从而导致合法连接建立失败。
		CrossPath 攻击	利用控制流量和数据流量之间的共享链路来拥塞控制信道。
控制通道	带宽资源	Packet-In 泛洪攻击	攻击者通过随机伪造数据包, 短时间内触发大规模的 Packet-In 消息在控制通道中传输, 占据大量带宽资源。
		Table-miss 增强攻击	一种隐蔽性好, 攻击效率高的特殊 Packet-In 泛洪攻击。
数据平面	计算资源	恶意规则注入攻击	攻击者迫使交换机安装大量虚假流规则, 导致计算资源的大量消耗。
		计数器操控攻击	一种更加隐蔽, 更加精细的恶意规则注入攻击。
	内存资源	流表溢出攻击	大量的恶意流规则占据有限的流表空间, 导致合法规则无法安装。

OpenFlow 交换机将未能与现有流规则匹配的数据包通过 Packet-In 消息发送给控制器, 然后控制器根据自身的控制逻辑产生流规则, 并下发至交换机。攻击者通过伪造虚假数据包迫使交换机短时间内向控制器发送大量 Packet-In 消息。控制器耗尽可用的计算资源来处理这些无效的 Packet-In 消息, 从而导致无空闲资源来处理合法的 Packet-In 消息[53]。图 3 显示了 Packet-In 泛洪攻击场景, 恶意主机不停地向交换机注入攻击流量, 致使整个网络崩溃。

我们在实验平台上重现了 Packet-In 泛洪攻击, 攻击效果如图 4 所示。当不存在攻击时, 即网络中仅存在正常流量, 控制器 CPU 平均利用率大约为 25%; 当恶意主机注入攻击流量时(10 秒处), 控制器 CPU 利用率急速上升至大约 90%。图 5 展示了当发生

Packet-In 泛洪攻击时, 合法主机之间的通信时延会大大增加, 平均通信时延由攻击前的 7 毫秒增加至攻击发生后的 210 毫秒。

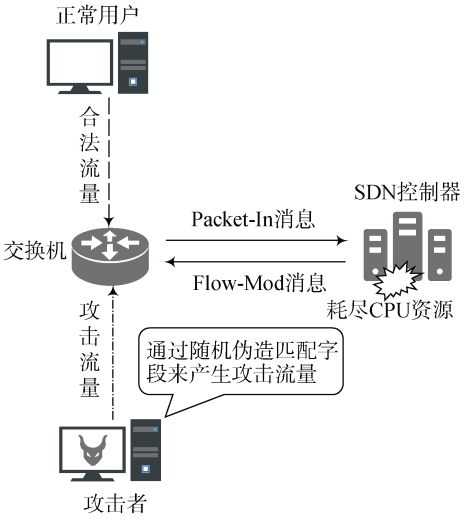


图 3 Packet-In 泛洪攻击过程

Figure 3 Process of Packet-In attack

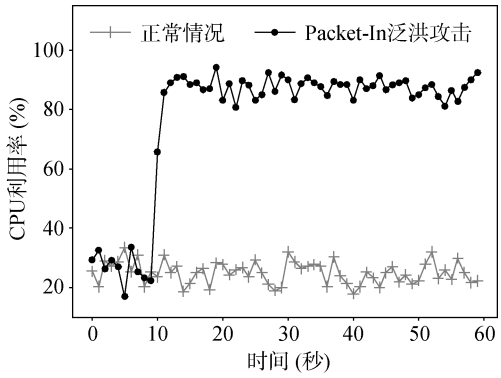


图 4 Packet-In 泛洪攻击对控制器 CPU 的影响

Figure 4 Impact of Packet-In attack on controller CPU

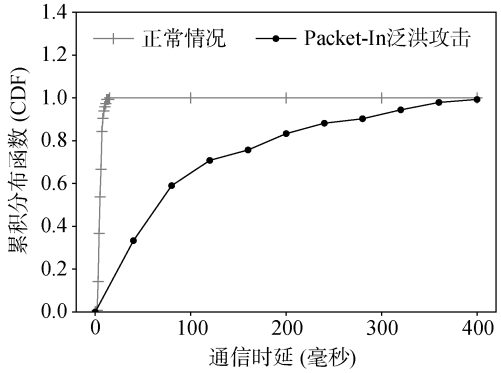


图 5 Packet-In 泛洪攻击对主机通信时延的影响

Figure 5 Impact of Packet-In attack on host communication delay

4.1.2 Table-miss 增强攻击

攻击者能力. 攻击者在基于 SDN 的网络中拥有一个或多个傀儡主机(例如通过恶意软件感染), 并且攻击者可以利用这些主机发送攻击数据包并获得其通信时延。攻击者无需利用漏洞直接破坏控制器, 应用程序或交换机的功能。同时应用程序和控制器之间以及控制器和交换机之间的连接受到安全协议(如 HTTPS 和 SSL/TLS)的良好保护。

Table-miss 增强攻击是一种更加复杂、更加隐蔽的 Packet-In 泛洪攻击[56]。与普通 Packet-In 泛洪攻击中随机伪造攻击包的方式不同, Table-miss 增强攻击首先探测出数据包首部中的敏感字段, 然后采用精确、低成本的攻击方式来消耗网络资源。如图 6 所示, Table-miss 增强攻击包括两个阶段: 探测和触发。在探测阶段中, 攻击者窃取数据包首部的敏感字段, 这些字段触发 SDN 应用程序产生流规则。然后攻击者根据探测到的敏感字段精心设计攻击数据包, 使得每一个攻击包都可以精确地触发交换机发送 Packet-In 消息至控制器。

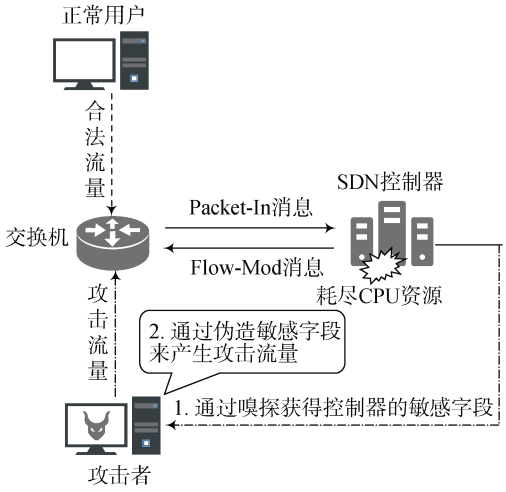


图 6 Table-miss 增强攻击过程

Figure 6 Process of table-miss striking attack

探测阶段是整个攻击的关键步骤。攻击者向网络发送首部字段被随机伪造值填充的探测包并观察其响应时间, 即探测包的往返时间(Round-Trip Time, RTT)。如果具有相同首部的多个数据包获得了不同的 RTT, 尤其第一个数据包经历了长时间延迟, 而其它数据包获得相对快速的响应, 攻击者可以推测出第一个数据包被定向到控制器而其它数据包直接在数据平面上完成转发, 这表示该数据包没有匹配交换机中现有的流规则, 并触发了 Packet-In 消息。随后, 攻击者更改探测包首部的任一字段来构造新探测包, 如果新探测包没有引入显著的 RTT 差异, 则

表明更改的字段没有触发 SDN 控制平面产生新规则, 即它是一个不敏感字段。由于最新 OpenFlow 规范[57] 仅支持 42 个数据包首部字段, 因此不超过 42 次试验, 攻击者便能够确定哪些首部字段是控制器的敏感字段。最后, 攻击者随机伪造探测阶段所获得的敏感字段来实现精确攻击, 从而获得比传统 Packet-In 泛洪攻击更加明显的攻击效果。

图 7 证明了 Table-miss 增强攻击比 Packet-In 泛洪攻击更加有效, 即在相同攻击速率下, 增强攻击会导致更多控制器计算资源的消耗。由于准确地推断首部敏感字段, 攻击者仅在敏感首部空间中变化字段值, 以此来制作攻击数据包, 从而使得每个攻击数据包都会触发 Packet-In 消息, 而传统 Packet-In 泛洪攻击采用随机伪造任意字段值来构造攻击数据包, 这显然无法保证攻击的效率。

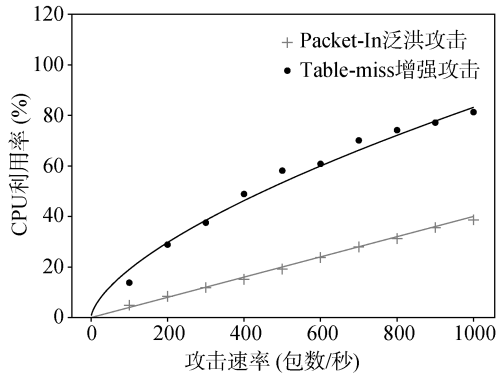


图 7 Table-miss 增强攻击对控制器 CPU 的影响

Figure 7 Impact of striking attack on controller CPU

4.1.3 恶意应用资源消耗攻击

攻击者能力. 假定 SDN 控制器是受信任的并且有足够的安全性。攻击者制作恶意的 SDN 应用程序, 并将其发布到第三方应用商店, 然后通过伪造数字签名等手段使得该恶意应用成功绕过安全验证并成功部署到 SDN 控制器中。

为了扩展控制平面功能, 网络服务通常会以 SDN 应用程序的方式部署在控制器上。与移动应用生态系统(如 Google Play 或 Apple App Store)类似, SDN 应用程序可以来自第三方应用程序商店, 这无疑加快了网络服务的创新进程, 如惠普公司已经建立了一个 SDN 应用商店[58]。同时许多 SDN 控制器供应商发布了应用程序接口来促进 SDN 应用程序的开发。目前 SDN 应用程序运行在控制器实例中, 就像传统应用运行在操作系统中, 因此具有不同处理逻辑的应用会造成不同的控制器计算资源消耗。攻击者可以将包含恶意处理逻辑(如恶意循环)的 SDN 应用程序部署到控制器中, 从而在运行过程中耗尽控制器的计算资源。我们通过在 Ryu 控制器上部署多种的应用程序来展示处理逻辑复杂性与计算资源消耗之间的关系。在实验中, 恶意应用是通过向正常应用中注入恶意循环代码所实现的, 图 8 展示了恶意应用会消耗更多的控制器计算资源。

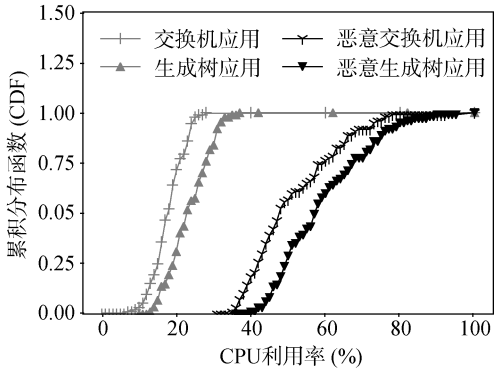


图 8 恶意应用对控制器 CPU 的影响

Figure 8 Impact of malicious applications on controller CPU

同消耗计算资源的恶意应用类似, 由于不规范的编程习惯或者恶意代码注入, SDN 应用程序可以使用多种方式来消耗控制器的内存资源。例如在每一次接收到交换机的请求消息时, 应用程序会创建新的变量来保存数据包字段信息, 但是在决策产生之后, 应用程序并没有及时撤销变量并释放内存资源, 从而导致随着接收到越来越多的请求报文, 控制器内存资源的消耗持续增加, 直至全部耗尽。

4.1.4 交换机表溢出攻击

攻击者能力. 在 SDN 控制器与交换机之间仅建立 TCP 连接时(为了保证网络性能, 网络管理员会禁止使用 SSL/TLS 安全连接[12]), 攻击者通过脚本程序来模拟交换机与 SDN 控制器建立连接, 然后攻击者通过静态分析的手段来识别 FEATURES_REPLY 消息, 并通过随机该消息中的交换机标识字段来伪造大量交换机与 SDN 控制器建立连接。

根据 Dover[59]的分析, 攻击者通过持续发送虚假的 OpenFlow 消息会造成 Floodlight 控制器中维护的交换机表溢出, 从而导致控制器与交换机之间的合法连接断开。具体地, 当 Floodlight 接收到具有新 DPID 值的 Features_Reply 消息时, 它会在交换机表中新增一个交换机条目。因此攻击者可以通过连续注入具有不同 DPID 值的 Features_Reply 消息来无限制地填充交换机表。该攻击将导致控制器中可用的内存资源耗尽, 最终迫使控制器断开与交换机之间

的合法连接。文献[59]中的实验结果所示, 交换机表溢出攻击不会立即产生影响, 而是持续了约 90 分钟耗尽 2GB RAM, 最终使得控制器完全崩溃。

4.2 控制通道攻击

在 SDN 网络中, 控制平面和数据平面是解耦的, 逻辑集中的控制器通过建立在南向协议(如 OpenFlow)上的控制通道与交换机进行通信。尽管 TLS/SSL 安全协议能够保证控制平面和数据平面之间传输的控制消息的机密性和完整性, 但是控制通道的可用性仍然可能遭受攻击者的破坏。攻击者通过消耗控制通道的带宽资源能够明显降低控制平面和数据平面之间的通信性能, 甚至完全阻塞控制器与交换机之间的控制通道连接。

4.2.1 CrossPath 攻击

攻击者能力. 在使用 OpenFlow 协议部署的 SDN 网络中, 控制器采用 In-band 控制信道来安装流规则。攻击者不需要破坏控制器, 应用程序和交换机, 也不需要在控制通道上构造中间人攻击来操纵控制消息, 同时攻击者无需具备直接进行网络操作的特权, 而仅仅是通过恶意软件等方式入侵 SDN 网络中的至少一台主机。攻击者的目标是利用入侵主机制作数据流量, 以破坏传输控制流量的 SDN 控制通道。

目前 SDN 网络中控制器与交换机之间的控制通道有两种实现方式: Out-of-band 和 In-band。如图 9 所示, Out-of-band 方式是指在控制器和交换机之间部署专用链路来传输控制消息, 如控制器与转发设备 D 之间通过专用链路 L 进行通信。这种方式会带来非常高的建设成本和管理成本, 因此该方式通常不会被使用, 特别是在大型网络中(如 SD-WAN)。In-band 方式是指控制器与交换机之间的控制流量与交换机之间的数据流量共享底层的数据链路, 如控制器与转发设备 D 通过数据链路 $l1$ 与 $l2$ 进行通信。如今 In-band 方式是控制流量通信的最佳选择, 然而这种方式同时也会给 SDN 网络引入新的安全威胁。攻击者向共享链路注入恶意数据流量来破坏控制通道连接的可用性, 这种攻击被称为 CrossPath 攻击[60]。在 CrossPath 攻击中, 攻击者为共享链路精心设计数据流量, 使得共享链路快速拥塞, 隐式地干扰控制流量的交付, 从而导致 SDN 控制器和交换机之间传递的实时性控制消息被延迟或丢弃。由于 SDN 控制器通过控制通道对分布式交换机执行集中控制功能, 攻击者利用 CrossPath 攻击可以轻易地影响控制器上运行的各种网络服务。

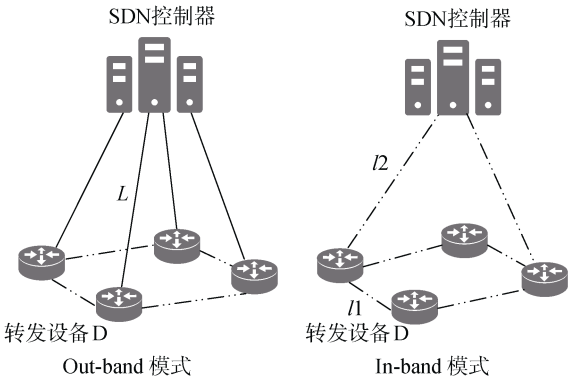


图 9 Out-of-band 和 In-band 部署模式

Figure 9 Out-of-band and in-band deployment patterns

实现 CrossPath 攻击的一个关键挑战是如何发现控制流量和数据流量的共享链路。传统 IP 网络中几乎所有的链路都同时传输控制流量(如 OSPF 或 BGP 更新)和数据流量, 而 SDN 网络中只有少量的链路转发控制流量。因此攻击者首先要在诸多的转发路径中发现一条包含共享链路的目标路径, 这在网络拓扑和路由信息对终端用户不可见的条件下是很难实现的。文献[60]提出了一种称为对抗性路径侦察的探测技术来发现与控制路径共享链路资源的目标数据路径。具体地, 该探测技术包括两个步骤: 控制路径时延探测和目标数据路径识别。其中控制路径时延为无规则匹配时探测包的转发时延与有规则匹配时探测包转发时延之差。攻击者然后注入测试数据流量, 如果短时间内暴增的数据流量导致控制路径上消息的传输延迟显著增高, 则说明该数据路径与控制路径存在共享链路, 据此攻击者通过探测控制消息的时延变化(注入/不注入短期突发数据流量)来识别与控制路径包含共享链路的目标数据路径。最后, 攻击者利用 TCP 超时重传机制的低效性, 通过周期性地发送脉冲数据流来发起低速拒绝服务攻击, 使得交换机发出的控制报文频繁发生超时重传, 进而更加高效, 更加隐蔽地实现 CrossPath 攻击。

我们在实验平台上测试了 CrossPath 攻击对 SDN 基本转发服务的影响。基本转发服务首先解析无法匹配任何流规则的数据包, 并查找目的 MAC 地址和交换机端口之间的映射记录。如果目标 MAC 地址已经与某个端口关联, 则数据包将被发送到该端口, 控制器下发相应的流规则来指导后续数据包的转发, 否则数据包将被泛洪至所有端口。CrossPath 攻击通过干扰控制器和交换机之间的通道, 有效降低转发决策的安装效率。图 10 展示了 CrossPath 攻击对交换机与控制器之间控制消息吞

吐量的影响。攻击发生前, 控制消息吞吐量约 1500 pps (Packets Per Second), 而攻击发生后, 吞吐量降低为约 120 pps。

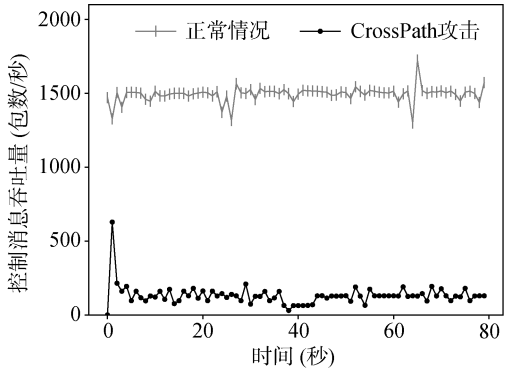


图 10 CrossPath 攻击对控制通道的影响
Figure 10 Impact of CrossPath attack on control channel

4.2.2 Packet-In 泛洪攻击

SDN 控制平面的集中式设计使得控制通道成为一个被频繁使用的中间媒介。控制通道中传输流量的一个重要原因是管理流规则。在 OpenFlow 中, 用于管理流规则的控制流量包括 Packet-In 消息和 Flow-Mod 消息。DevoFlow[61]报告指出在包含 N 个交换机的转发路径上为一个新数据流进行单向规则安装会产生 94 + 144N 字节(1 个 Packet-In 消息以及 N 个 Flow-Mod 消息)的控制流量。因此, 当攻击者发起 Packet-In 泛洪攻击时, 新数据流以高速率进入交换机, 短时间内触发大量的 Packet-In 消息和 Flow-Mod 消息在控制通道中传输, 最终导致控制通道的带宽资源被严重消耗。据 DevoFlow 测量, HP ProCurve 5406zl 型号的交换机仅仅可以支持 17 Mbps 大小的控制通道带宽, 攻击者可以轻易地耗尽带宽资源, 阻断交换机和控制器之间的合法通信。

我们在实验平台上测量了 Packet-In 泛洪攻击对控制通道带宽的影响。如图 11 显示, 当网络中仅传输合法流量时, 控制消息大约消耗了约 500 Kbs 带宽资源, 当主机以相同的速率向网络中注入攻击流量时, 控制通道的带宽消耗会上升至约 1500 Kbs。随着恶意主机数量的增多以及攻击速率的升高, 控制通道会被恶意控制消息完全阻塞, 从而导致合法主机之间的通信无法正常完成。

4.2.3 Table-miss 增强攻击

根据 4.1.2 小节描述, Table-miss 增强攻击是一种更加有效、更加隐蔽的 Packet-In 泛洪攻击。它可以在相同的攻击速率下触发更多的 Packet-In 消息和 Flow-Mod 消息在控制通道中传输, 从而消耗更多的

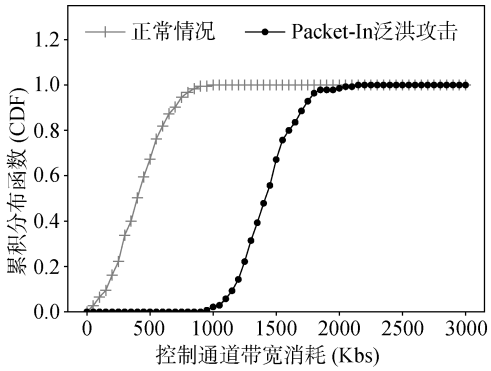


图 11 Packet-In 泛洪攻击对控制通道带宽的影响
Figure 11 Impact of Packet-In attack on control channel bandwidth

控制通道带宽资源。我们在实验平台上比较了 Table-miss 增强攻击和普通 Packet-In 泛洪攻击对控制通道带宽资源的消耗情况。图 12 直观地展示了 Table-miss 增强攻击对控制通道带宽资源造成更严重的消耗。假如 SDN 网络中存在 1000 个攻击速率为 1000 pps 的恶意主机, Table-miss 增强攻击大约可以消耗 2.5 Gbs 的带宽资源, 而普通 Packet-In 泛洪攻击仅仅消耗了 0.7 Gbs 的带宽资源。

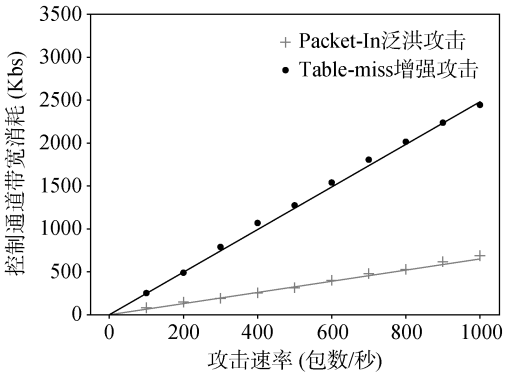


图 12 Table-miss 增强攻击对控制通道带宽的影响
Figure 12 Impact of Table-miss striking attack on control channel bandwidth

4.3 数据平面攻击

当前商用 SDN 硬件交换机(如 IBM RackSwitch、Juniper Junos max -Series、Brocade NetIron CES 2000 Series、Pica8 Series 以及 Hewlett-Packard Series 等硬件交换机)的处理能力受到多种因素的制约。第一, 由于成本原因, 硬件交换机的 CPU 性能通常比较弱, 这限制了交换机中软件协议代理的消息解析和处理能力。第二, 大多数商用硬件交换机使用三态内容寻址存储器(Ternary Content Addressable Memory, TCAM)保存流规则, 以实现线速数据包处理。但是由于高成本和高功耗的原因, TCAM 只允许有限的流

表更新率(仅支持每秒 100 至 200 流规则更新[62])和小流表空间(从几百到几千[61])。攻击者能够采用多种攻击手段消耗 SDN 交换机中有限的计算资源和流表资源, 使其丧失正常工作的能力。

4.3.1 恶意规则注入攻击

攻击者能力. 当攻击者拥有如下两种能力之一时便能够实现规则注入攻击: 1、攻击者有能力将恶意应用程序部署到 SDN 控制程序中(例如通过伪造数字签名的方式), 因此当接收到交换机消息时, 恶意程序可以下发大量流规则到交换机; 2、攻击者通过中间人攻击伪造控制器向交换机下发流规则。

由于缺乏认证机制, SDN 交换机对来自控制器一侧的流规则没有任何限制地进行解析和安装, 因此攻击者可以通过伪造包含无效流规则的 Flow-Mod 消息或者部署一个恶意 SDN 应用程序来向交换机不断地下发流规则。解析和处理大量的 Flow-Mod 消息会占用甚至耗尽交换机贫乏的 CPU 资源, 从而降低交换机对其它合法消息的处理性能。

图 13 显示了恶意规则注入攻击前后网络延迟的巨大变化。在攻击发生前, 合法主机之间的通信时延约为 9 毫秒, 而当恶意规则以每秒 1000 流规则的速率注入交换机时, 通信时延增加至约 138 毫秒。

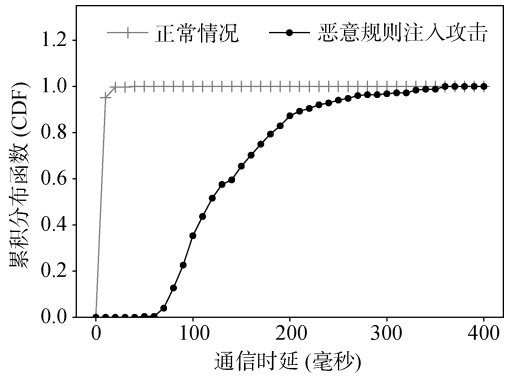


图 13 恶意规则注入攻击的攻击效果

Figure 13 Attack effect of malicious rule injection

4.3.2 计时器操纵攻击

攻击者能力. 在控制器、应用程序以及交换机保护良好的情况下, 攻击者仅通过控制 SDN 网络中的一个或多个傀儡主机, 并利用这些主机发送探测报文、监听其转发时延以及发送攻击流量便可以实现计时器操作攻击。

计数器操控攻击是一种更加精细的恶意规则注入攻击。它不需要攻击者向控制器部署经过设计的恶意应用或伪造控制消息, 而是利用合法应用的正常逻辑发起的一种更加隐蔽且更易实施的攻击。

在 OpenFlow 协议中, 每一个流表项都包含多个计数器来记录已匹配某一流规则的数据包个数、字节数等统计信息。SDN 应用程序可以利用 Statistics Query/Reply 控制消息来获得这些计数器的值, 并据此进一步实现优化、监控以及保护网络的目标。计数器操纵攻击的可行性依赖于不同类型的下行控制消息对交换机计算资源的消耗不同。在 SDN 应用程序与数据平面的交互过程中, 下行控制消息主要包含三种类型, 即 Flow-Mod 消息、Statistics Query 消息和 Packet-Out 消息。其中交换机处理 Flow-Mod 消息的计算资源开销最大, 因为它不仅消耗交换机协议代理的 CPU 资源来解析消息, 而且还涉及到调用 API 来插入或者修改流规则, 其次是 Statistics Query 消息, 它既需要交换机协议代理的 CPU 资源进行数据包解析, 也需要调用 API 进行统计查询。相比较而言, Packet-Out 消息是相当轻量级的, 因为它只涉及到使用交换机协议代理的 CPU 资源来执行消息中封装的相应操作[56]。由于这三种类型的下行消息会给交换机带来不同的计算负载, 所以当交换机接收到不同类型的控制消息时, 处理其他数据包的时延会有所不同。攻击者根据探测包转发时延来判断控制平面发出的是 Flow-Mod 消息, Statistics Query 消息, 或者 Packet-Out 消息, 从而准确地推测 SDN 应用程序中 Statistics Query/Reply 消息的使用逻辑。一般来说, SDN 应用程序会定期使用 Statistics Query/Reply 消息来获得交换机的统计信息, 每一个查询消息都会导致探测包转发时延的增加, 这个增加的时延揭示了应用程序查询统计信息的周期。如果控制器后续发出 Flow-Mod 消息, 则探测包转发时延会再次增加。基于这种特殊的现象, 攻击者甚至可以推断出 SDN 应用程序所采用的流规则计算方法, 如基于字节数下发流规则或基于数据包速率下发流规则。当获得统计信息查询周期以及触发下行控制消息产生的逻辑条件等机密信息后, 攻击者通过精心设计每个数据流中的报文间隔以及报文大小, 使得计时器在每一个查询周期内都能达到临界值, 从而精准地触发控制器下发控制信息, 最终耗尽交换机的计算资源。

文献[56]的实验结果验证了计数器操控攻击的可行性。攻击者通过比较探测包的转发时延来确定 SDN 应用程序的统计信息查询周期。图 14 所示每隔 2 秒, 探测包的转发时延就发生一次增加, 因此推测应用程序的统计信息查询周期为 2 秒。通过进一步构造攻击流量, 攻击者迫使控制器在一个查询周期内除下发统计信息查询消息外, 还要下发 Flow-Mod

消息, 因此造成了如图 15 所示的“双峰”现象。

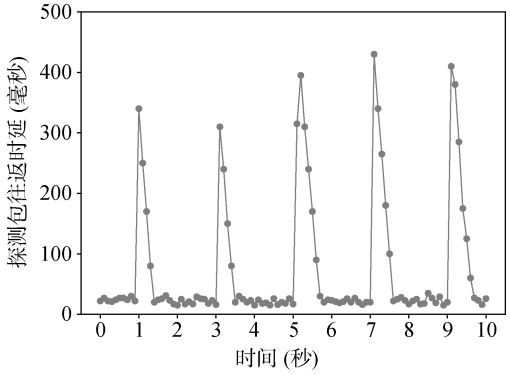


图 14　CrossPath 攻击窃取统计信息查询周期的过程
Figure 14　Stealing statistics query period via CrossPath

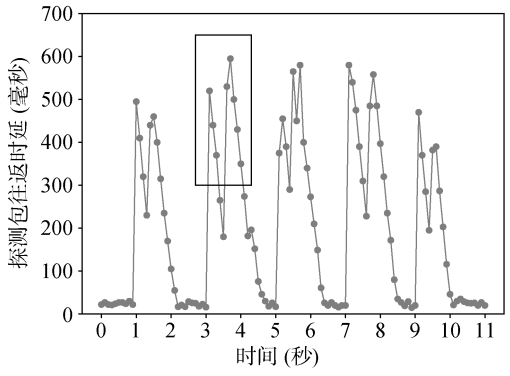


图 15　CrossPath 中的“双峰”现象
Figure 15　Double peak in CrossPath attack

4.3.3　流表溢出攻击

攻击者能力. 假设 SDN 网络中的控制器、交换机、应用程序都得到了良好的保护, 并且南向接口和北向接口也被安全协议所保护, 如 SSL/TLS 和 HTTPS, 攻击者仅仅需要入侵 SDN 网络中的一个或者多个主机, 并利用这些入侵主机通过地址欺骗来伪造大量的虚假数据包, 以触发控制器向交换机下发大量的流规则。

目前SDN商用交换机广泛使用TCAM作为流表来存储流规则。TCAM 的优点是支持并行访问存储条目, 查询速度快且几乎不受存储条目数量的影响。但由于高成本与高能耗的原因, 大部分 SDN 商用交换机仅仅可以存储数以千计的流规则, 如 Pica 8 商用交换机只能够存放 8192 条流规则[63], 这对于大规模网络是远远不够的。当交换机流表没有足够的空间安装新的流规则时, 交换机会直接丢弃新规则, 这将导致数据包无法正常转发。针对流表空间有限的脆弱性, 攻击者可以在短时间内生成大量随机报文, 迫使控制器集中地向交换机下发流规则。由随机报文产生的恶意流规则很快就会耗尽流表资源, 使得合法流规则无法安装, 最终导致网络服务不能正常提供, 这种攻击方式被称为流表溢出攻击。该攻击不需要攻击者具有控制平面的访问权限, 也不需要攻击者挖掘控制器或交换机中的任何软件漏洞, 大大降低了攻击门槛。此外, 由于新兴的可编程数据平面(如 P4 和 RMT 芯片[64])同样使用 TCAM 作为流表存储流规则, 因此它们同样面临着遭受流表溢出攻击的风险。流表溢出攻击与恶意规则注入攻击以及计时器操纵攻击存在一定的差异。具体地, 流表溢出攻击是通过在流表中新增流规则来耗尽匮乏的 TCAM 资源, 而恶意规则注入攻击和计时器操纵攻击可以表现为向流表中注入新规则, 也可以是攻击者频繁地更新现有规则而不造成TCAM资源的消耗, 因此其攻击目标是消耗交换机的计算资源。

我们在实验平台上测试了流表溢出攻击对流表资源和合法数据包转发时延的影响, 实验结果见图 16 和图 17。在实验中, 交换机流表被设置为最多可安装 2000 条流规则, 以及每一条流规则的生存周期为 10 秒。当网络中仅存在合法流量时, 交换机流表大约存在 250 条流规则, 合法主机之间的平均通信时延为 7 毫秒; 当攻击者发起流表溢出攻击后, 流表中的规则数量急剧上升, 直至流表被填充满, 此时合法主机之间的通信时延上升至约 688 毫秒。

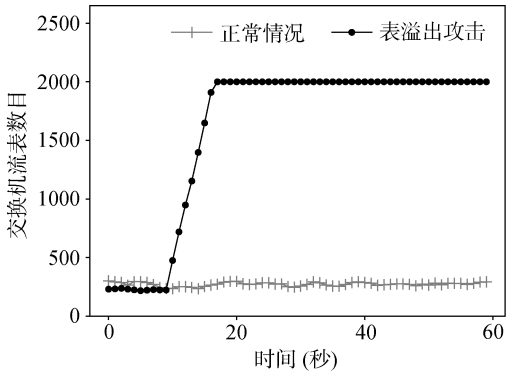


图 16　流表溢出攻击对流表资源的影响
Figure 16　Impact of table-overflow attack on flow table resources

5　现有防御机制

随着诸多 SDN 应用场景的不断提出, 其网络架构的安全问题引起了学术界和工业界的广泛关注。如表 3 所示, 当攻击者仅仅具有入侵 SDN 网络中主机的能力时, 如根据操作系统中的漏洞获得主机的

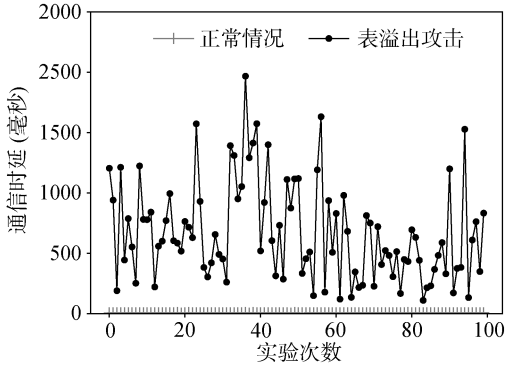


图 17 流表溢出攻击对通信时延的影响

Figure 17 Impact of table-overflow attack on communication delay

表 3 攻击者能力与可实现攻击类型的关系

Table 3 Relationship between attacker capabilities and types of implementable attacks

攻击者能力	可实现攻击类型	主要受影响的资源类型
控制网络主机	Packet-In 泛洪攻击	控制平面的计算资源
	Table-miss 增强攻击	控制平面的计算资源
	CrossPath 攻击	控制通道的带宽资源
	计数器操控攻击	数据平面的计算资源
	流表溢出攻击	数据平面的内存资源
部署恶意应用	恶意应用资源消耗攻击	控制平面的计算资源和内存资源
	恶意规则注入攻击	数据平面的计算资源
窃听控制消息	交换机表溢出攻击	数据平面的内存资源
	恶意规则注入攻击	数据平面的计算资源

控制权, 便能够实施多种资源消耗型攻击。本节对防御上述资源消耗型攻击的现有研究成果进行了归纳和分析。

5.1 控制平面

Packet-In 泛洪攻击、Table-miss 增强攻击、交换机表溢出攻击以及恶意应用资源耗尽攻击会对控制器的计算资源或内存资源造成极大的消耗。接下来, 本小节介绍了针对这些攻击的现有防御机制。

5.1.1 Packet-In 泛洪攻击的防御机制

对于 Packet-In 泛洪攻击, 现有缓解方案可大致分为三种: 基于流量特征的检测机制、基于流量迁移的防御机制以及基于分布式控制器的缓解机制。表 4 总结了针对 Packet-In 泛洪攻击的多种防御机制。

Packet-In 泛洪攻击的一个明显特征是短时间内无法与流表匹配的流量大幅度增加, 因此基于流量特征变化来检测 Packet-In 泛洪攻击是一种常用的方法。Wang 等人[65-70]提出了基于熵的轻量级攻击检测机制, 控制器根据流量随机性来检测网络中是否发生攻击。当大量随机数据包向受害主机发送时, 未匹配的数据包会被发送至控制器, 控制器通过计算检测窗口内记录的数据包熵, 并通过与正常状况下的阈值进行比较来判断是否发生了攻击(若发生攻击, 数据包目的 IP 地址的熵通常比正常状况小, 而源 IP 地址的熵会比正常状况大)。此外, 研究人员还可以利用 SDN 控制器集中管控的特点, 周期性地采集流表统计信息, 然后利用机器学习或神经网络等算法来检测攻击的发生, 如 Braga 等[71]提取了流表的六个特征(信息流中数据包的平均数量、信息流的平均字节数、信息流的平均持续时间、配对信息流的百分比、单信息流的增长率以及不同端口的增长率), 然后利用自组织映射算法(Self Organizing Maps, SOM)对 Packet-In 泛洪攻击进行检测, 如图 18 所示, 该方式无需修改 OpenFlow 协议或者基础设施, 具有很高的通用性。

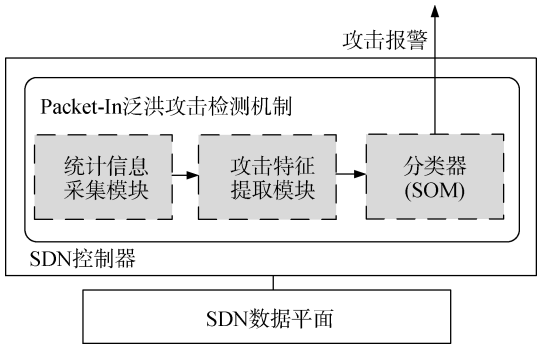


图 18 基于 SOM 算法的 Packet-In 泛洪攻击检测机制

Figure 18 Packet-In flooding attack detection mechanism based on SOM algorithm

基于流量迁移防御机制的原理是在 SDN 控制器和交换机通信过程中增加一个过滤模块, 对请求信息进行检测和过滤。过滤模块丢弃或转移非法或虚假的请求报文, 仅将合法的请求报文发送至控制器。如图 19 所示, Avant-Guard[31]在数据面已有的数据包转发和统计信息采集功能的基础之上, 新增了连接迁移模块。该模块对 TCP 连接的三次握手进行验证, 具体过程如下: 当交换机收到 TCP SYN 报文时, 连接迁移模块代替目的主机向源主机回复 TCP ACK 报文, 当连接迁移模块再次收到了对应 TCP 会话中的 SYN ACK 报文, 此 TCP 会话将会被判定为正常的 TCP 会话, 然后该模块便会将属于该会话的后续数据包迁移至 SDN 控制器。通过添加连接迁移模块, 数据平面实现了对 TCP 数据包的检测和过滤, 丢弃了仅仅建立 TCP 半连接的恶意数据包。Afek 等人[72]

按照 OpenFlow 1.5 协议在 Open vSwitch 和 P4 交换机上实现了 Avant-Guard 的迁移思想。但是 Avant-Guard 本身可能会引发一些新的安全问题, 如 Avant-Guard 作为一个代理需要处理来自主机的所有 TCP 连接, 因此它必须在交换机内存中为每个连接都维护一些状态信息, 这无疑增加了交换机内存溢出的风险。此外, Avant-Guard 可代理的最大连接数受限于当前可用的 TCP 端口数量。为解决上述问题, Ambrosin 等人[34]提出了一个新型的流过滤模块, 称为 LineSwitch。它使用网络流量的概率代理和黑名单机制来防止攻击流量到达控制平面。然而无论是 Avant-Guard, 还是 LineSwitch, 它们仅适用于 TCP 攻击流。为了弥补支持协议上的不足, Wang 等人[32]提出了可以检测和防御多种协议攻击的 FloodGuard 架构, 它在 SDN 原有的基础设施之外引入了数据面缓存模块, 并结合主动流规则分析技术来推断当前网络所需要的流规则。当交换机接收到的数据包与流表规则无法匹配时, 该数据包会被重新引导至数据面缓存, 然后数据面缓存模块将数据包以较低的速率发送至控制器。SDNShield[76]同样引入了一组额外的设备对进入 SDN 网络中的流量进行两段式过滤

表 4 Packet-In 泛洪攻击的缓解方案

Table 4 Mitigation of Packet-In flooding attack

防御原理	具体方案
基于流量特征的检测机制	Wang et al. (2015) [65] Huang et al. (2017) [66] Dong et al. (2016) [67] Cui et al. (2019) [68] JESS (2018) [69] DosDefender (2019) [70] Braga et al. (2010) [71] SDN-Guard (2016) [73] Sguard (2017) [74] Lapolli et al. (2019) [75]
基于流量迁移的防御机制	Avant-Guard (2013) [31] FloodGuard (2015) [32] FloodDefender (2017) [33] LineSwitch (2017) [34] Afek et al. (2017) [72] SDNShield (2016) [76] PacketChecker (2018) [77] CoFilter (2018) [78]
基于分布式控制器的缓解机制	Onix (2010) [38] Ravana (2015) [39] HyperFlow (2010) [79] Kandoo (2012) [80] Vyatta Controller (2014) [81] ONOS (2015) [82] SCL (2017) [83] OpenDaylight (2015) [84] Open SDN Controller (2017) [85] Tungsten Fabric(2018) [86] Contrail (2015) [87]

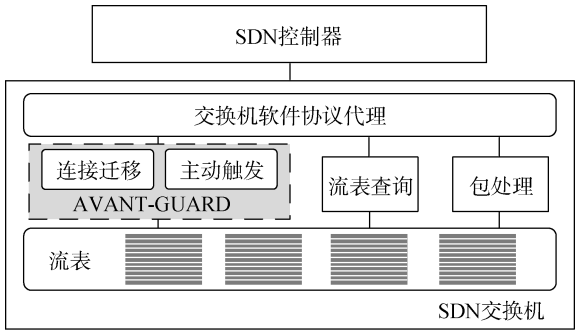


图 19 Avant-Guard 系统架构

Figure 19 Architecture of Avant-Guard

(统计信息和 TCP 会话检测), 避免了恶意流量进入网络。然而 FloodGuard 和 SDNShield 机制都需要引入额外设备来暂时存储攻击流量。为了增加防御机制的通用性, Gao 等人[33]将基于频率的过滤机制移植到控制器并且设计了 FloodDefender 攻击防御系统。FloodDefender 以 SDN 应用程序的形式部署在网络中, 它无需修改 SDN 协议和添加基础设施。同样地, PacketChecker[77]作为控制器的轻量级扩展, 在控制器核心模块处理请求消息之前, 根据数据包首部字段对 Packet-In 消息进行过滤。通过比较 MAC 地址、交换机 DPID 和端口信息, PacketChecker 自动识别恶意数据包并通过 Flow-Mod 消息通知交换机丢弃这些报文, 避免了控制器计算资源的消耗。表 5 总结了上述两种机制中典型防御方案的特点与不足。

分布式控制器架构的出现极大地增加了 SDN 控制平面的可扩展性。在 SDN 架构中, 集中式控制平面具体指的是逻辑集中的控制平面。事实上, 为了满足性能、可伸缩性和可靠性要求, 生产级 SDN 网络通常采用物理分布式的控制平面。当采用分布式控制平面时, 来自数据平面的 Packet-In 请求将会按照负载均衡等机制发往不同的控制器实例, 从而避免某一个控制器实例的资源被过度消耗。OpenFlow 协议从 1.2 版本开始便支持多控制器部署。交换机与多个控制器建立连接(主/从连接), 并规定只有建立主连接的控制器可以对其进行控制操作。当主控制器发生故障时, 某一从控制器根据控制器选择算法竞选成为新的主控制器。当前开放网络操作系统(Open Network Operating System, ONOS)[82] 以及 OpenDaylight(ODL)[84]是网络运营商和厂商使用最多的分布式控制器平台。ONOS 解决了 SDN 控制器性能瓶颈和单一故障点的问题。如图 20 所示, 一个大规模的广域网被划分为由不同 ONOS 控制器实例控制的多个网络域, 这些分布式控制器实例为网络运维人员构造了一个全局网络视图。ODL 是 Linux

基金会的一个协作项目, 旨在促进 SDN 的发展和使用, 目前 ODL 社区已经联合建立了一个开放的分布式控制器框架。此外, Onix[38]、Ravana[39]以及 HyperFlow[79]等分布式控制器架构同样增强了 SDN 控制平面的可扩展性和可靠性。表 6 对典型的分布式控制器做出了总结, 其中 Tungsten Fabric 控制器的前身是 Juniper 的 OpenContrail 控制器。为了使更多外部开发者参与到 OpenContrail 项目中, Juniper 公司于 2018 年将其托管至 Linux 基金会, 并重新命名为 Tungsten Fabric。

表 5 Packet-In 泛洪攻击防御机制对比

Table 5 Comparison of defense mechanisms against Packet-In flood attacks

方案名称	防御类型	是否增加或修改设备	是否适用全部攻击协议	是否会丢弃合法数据包	其他不足
SDN-Guard [73]	流量特征检测	否	是	是	丢失数据包统计信息以及只适用于多路径网络
SGuard [74]	流量特征检测	否	是	是	增加流时延
DosDefender [70]	流量特征检测	否	是	是	计算负载较大
Cui et al. [68]	流量特征检测	否	是	是	计算负载较大
Huang et al. [66]	流量特征检测	是	是	是	检测阈值难以确定
Avant-Guard [31]	流量迁移	是	仅 TCP 协议	否	存在交换机内存溢出的风险
PacketChecker [77]	流量迁移	是	是	否	增加流时延
FloodGuard [32]	流量迁移	是	是	否	增加流时延
LineSwitch [34]	流量迁移	是	仅 TCP 协议	否	增加流时延
Afek et al. [72]	流量迁移	否	TCP、DNS 协议	否	增加流时延
FloodDefender [33]	流量迁移	否	是	否	攻击检测算法简单
SDNShield [76]	流量迁移	是	仅 TCP 协议	否	增加流时延

表 6 分布式控制器的总结

Table 6 Summary of distributed controller platforms

控制器名称	所属组织	语言	开源	简介
Onix[38]	Google	C++	否	提供了通用分布式状态管理 API, 使控制面能够在全局视图下运行
Ravana [39]	普林斯顿大学	Python	否	提供了故障转移机制来解决分布式 SDN 控制器环境中的容错问题
HyperFlow [79]	多伦多大学	C++	否	一个基于事件的 OpenFlow 分布式控制平台, 可以实现多控制器之间协同工作
Kandoo [80]	多伦多大学	C++、Python	否	一种分层式的控制平面, 由本地控制器和根控制器组成, 其中本地控制器仅完成本地业务, 根控制器完成网络范围内的业务请求
Vyatta [81]	Brocade	Java	否	一款基于 OpenDaylight 的控制器, 并且支持 Brocade 的其他服务
ONOS [82]	ON.Lab	Java	是	专为可伸缩性和高可用性而设计的开源 SDN 控制器平台
SCL [83]	加利福尼亚大学	Python	否	基于单例控制器构建的分布式控制器框架, 通过改进一致性算法来提高整个网络的吞吐量
OpenDaylight [84]	Linux Foundation	Java	是	OpenDaylight 支持 OSGi 框架以实现本地控制器的可编程性, 以及双向 REST, 以实现作为北向 API 的远程可编程性
Open Controller [85]	Cisco	Java	否	基于 OpenDaylight 的 SDN 控制器, 具有其他 Cisco 嵌入式应用程序, 开发环境以及支持 Cisco MPLS 的 OpenFlow 协议扩展
Tungsten Fabric [86]	Linux Foundation	Python	是	Tungsten Fabric 是 Contrail 控制器的开源版本
Contrail [87]	Juniper	Python	否	一种旨在虚拟机上运行的软件控制器, 公开了一组 REST API, 用于与云编排工具以及其他应用程序进行北向交互

5.1.2 Table-miss 增强攻击的防御机制

Table-miss 增强攻击是一种特殊的 Packet-In 泛洪攻击, 但它在攻击方法和攻击效果上都有别于普通的 Packet-In 泛洪攻击。一方面, 普通的 Packet-In 泛洪攻击使用了一种非常简单的攻击方式, 即攻击者通过伪造任意首部字段来构造攻击流量, 触发大量恶意 Packet-In 消息进入控制器, 而在 Table-miss 增强攻击中, 攻击者通过嗅探技术获得 SDN 控制器的敏感字段, 然后仅仅通过伪造敏感字段来构造攻击流量。另一方面, 由于 Packet-In 泛洪攻击的简单性, 检测该类型攻击并不困难, 而 Table-miss 增强攻击具有更强的隐蔽性, 即攻击者只需发送低速攻击流量, 便可以实现显著的攻击效果, 这无疑增加了攻击检测的难度。

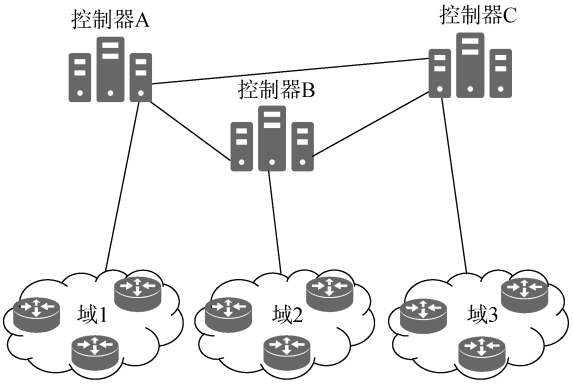


图 20 分布式控制器架构

Figure 20 Architecture of distributed controller

Table-miss 增强攻击中的一个关键过程是敏感字段的嗅探, 运维人员通过随机增加控制消息时延使流规则的产生策略难以被窃取, 从而避免攻击发生。然而这种技术增加了控制消息的总延迟, 不可避免地违反某些控制消息的延迟要求。Zhang 等人[56]提出了一个基于优先级的调度机制, 称为 SWGuard。如图 21 所示, 在交换机一侧, 现有的交换机软件代理增加了一个多队列缓存结构, 并根据不同协议类型和优先级调度控制消息。在控制器一侧, 增加了一个行为监控模块来收集不同的控制消息, 并基于一种新的监视粒度(Host-Application Pair, HAP)为控制消息动态分配优先级。在 SWGuard 中, 恶意控制消息更容易被放入低优先级调度队列中, 从而在资源紧张的情况下优先满足合法控制消息的时延要求。由于 SWGuard 系统区分了合法控制消息和恶意控制消息, 因此它同样可以缓解计时器操控攻击[56]。

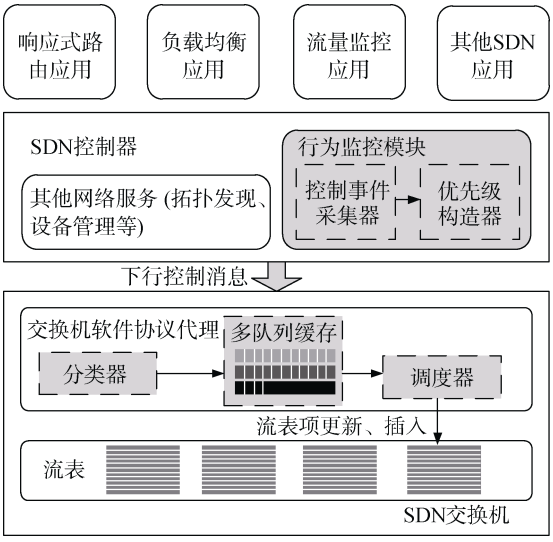


图 21 SWGuard 系统架构

Figure 21 Architecture of SWGuard

5.1.3 恶意应用资源消耗攻击的防御机制

在 SDN 中, 各种各样的网络功能是以 SDN 应用程序的形式实现, 因此保护 SDN 应用程序的安全性是网络功能正常运行的前提条件。为了避免恶意应用程序部署到 SDN 控制器上, 网络运维人员可以借鉴传统应用程序的静态分析和动态分析技术来检测出恶意 SDN 应用程序。此外, 针对 SDN 控制器中应用程序的安全性问题, Ball 等人[88]设计了工具 VeriCon 来验证 SDN 应用程序是否正常运行。Canini 等人[89]提出了 NICE 检测模型并使用该模型对真实的 Python 应用程序进行了漏洞测试和验证。Lee 等人[90]提出了一个新颖的 SDN 恶意应用检测系统, 称为 Indago, 它静态地分析 SDN 应用程序, 并对它们的行为概要进行建模, 最后使用机器学习方法自动检测出恶意 SDN 应用程序。

5.1.4 交换机表溢出攻击的防御机制

交换机表溢出攻击发生的原因是控制器缺乏对交换机身份的认证机制。当 SDN 控制器与交换机建立连接时, 控制器应该使用身份验证机制对每一个交换机进行身份验证, 从而避免恶意连接的建立。根据 OpenFlow 白皮书[57], 控制器与交换机之间的连接应建立在 TLS 安全协议之上, 从而可以利用权威中心签发的数据证书来实现控制器和交换机之间的身份认证。其次, 由于交换机表溢出攻击是由攻击者伪造虚假 Features_Reply 消息造成的(即通过随机生成消息中的交换机身份字段信息), 因此 Dover[59]建议可以根据控制器接收 Features_Reply 消息的时刻(合法的 Features_Reply 消息只会在控制器和交换机握手阶段出现)来丢弃攻击者或恶意交换机伪造的 Features_Reply 消息, 从而缓解对控制器的攻击。

5.2 控制通道

针对 Packet-In 泛洪攻击的防御机制可以在一定程度上缓解攻击流量对控制通道带宽资源的消耗, 即通过在网络入口交换机处阻止恶意流量进入网络来缓解控制消息对控制通道带宽的消耗。除此之外, 研究人员针对控制通道带宽资源还提出了多种保护机制[61,72,78,91-94], 如 DevoFlow[61]以及 DIFANE[91]。对相关文献进行归纳, 现有控制通道带宽资源缓解机制可大致分为如下两种: 控制逻辑卸载机制和优化控制信息传输机制。

5.2.1 控制逻辑卸载

控制层和应用层构成了 SDN 控制平面, 其中应用层可以部署各种网络应用程序, 以支持多种网络服务, 如路由、网络监视、异常检测和负载平衡。控制平面作为网络操作系统运行, 而数据平面则是由

功能极其简单的 SDN 交换机组成, 并根据控制平面生成的决策进行数据包处理和转发。由于控制平面和数据平面的完全解耦, 不可避免地引入过多控制信息(如流安装和统计信息查询等信息)在控制通道中传输, 消耗通道带宽资源。因此研究学者[61,72,78,91]提出了通过将部分控制逻辑转移到数据平面的交换机中来减少两个平面之间频繁的通信。DevoFlow[61]引入了“rule cloning”和“local actions”机制来实现向交换机转移部分控制权, 通过控制逻辑卸载, 使得交换机获得一定的决策能力, 从而交换机可以在不与控制器通信的前提下对部分数据流做出本地路由决策。DIFANE[91]引入了权威交换机并在权威交换机上预安装流规则, 以此代替控制器处理普通交换机的流规则安装请求, 从而将控制平面与数据平面的通信转变为数据平面内部的通信。Afek 等人[72]在交换机中实现了四种 SYN 反欺骗方法(TCP 代理、HTTP 重定向、TCP 安全重置以及 TCP 重置)和一种 DNS 反欺骗方法, 进而在数据包入口处过滤恶意流量。Cao 等人[78]利用 P4 原语设计了细粒度、轻量级和低延迟的实时攻击检测机制, 并将其在 P4 交换机中实现。此外, Avant-Guard[31]和 LineSwitch[34]在 SDN 交换机上扩展了流量过滤模块, 阻止了恶意流量进入 SDN 网络, 从而有效地避免了由恶意流量触发的控制消息在控制通道上传输。

5.2.2 优化控制消息传输

控制平面和数据平面之间传输的控制消息主要有两个用途: 流规则管理和统计信息采集。因此通过优化流规则管理机制和统计信息采集机制可以减轻控制通道带宽资源的消耗。

流规则的管理大多是通过流规则设置过程实现, 即交换机向控制器发送 Packet-In 消息, 控制器向交换机响应 Flow-Mod 消息。以安装流规则为例, 目前常见的流安装方式包括逐跳安装和一次性安装。假如采用逐跳安装流规则的方式, 那么对于包含 N 个交换机的转发路径, 完成流安装过程需要 N 个 Packet-In 消息和 N 个 Flow-Mod 消息。假如采用一次性安装方式, 只需要在数据包的入口交换机处发送一个 Packet-In 消息, 控制器同时向转发路径上的所有交换机发送 Flow-Mod 消息, 整个过程共需要一个 Packet-In 消息和 N 个 Flow-Mod 消息。Bu 等人[92]提出了一种新的流规则安装方式, 称为 FastLane, 其安装过程如图 22 所示。在 FastLane 中, 入口交换机在接收到无法匹配现有转发规则的数据包时, 向控制器发送一个 Packet-In 消息, 控制器将整个路径中所有交换机的转发动作封装到一个 Flow-Mod 消息中, 并将其发送至入口交换机, 然后入口交换机沿着数据链路同时转发包含处理规则的 Flow-Mod 消息和数据包。如此一来, 通过交换机之间的通信来完成整条路径上转发规则的安装, 减少了控制流量(一个 Packet-In 消息和一个 Flow-Mod 消息)在控制通道上的传输。通过文献[92]中的数学分析, FastLane 实现了比传统流安装方式更高的控制通道带宽效率, 特别是对于较长的转发路径, 如对于包含三个交换机的转发路径, FastLane 可以节省超过一半的控制通道带宽资源。Xu 等人[93]发现若考虑交换机处理转发规则和数据包之间的时延差异(交换机通过本地控制平面处理规则的时延比仅通过数据平面处理普通数据包的时延长得多), FastLane 可能会带来更大的控制通道带宽消耗和更高的传输时延, 因此改良了 FastLane 的设计并提出了 EFastLane 流规则安装机制。EFastLane 与 FastLane 不同之处在于没有明确指出要将整个路径的转发规则下发给入口交换机, 而是下发规则至转发路径上的某一个目标交换机, 然后目标交换机双向地向路径两端传递规则, EFastLane 流安装过程如图 23 所示。这样做的目的是保证交换机在转发数据包之前已经完成了相应流规则的安装, 从而切实保证在整个流规则安装过程中仅有一个 Packet-In 消息和一个 Flow-Mod 消息在控制通道上传输。此外, Li 等人[94]设计、实现并评估了 SDN 交换机中的 Packet-In 缓存系统。通过使用缓存系统, 只有数据流中第一个报文触发的 Packet-In 消息会被发送到控制器, 而后续报文触发的 Packet-In 消息会被临时存储在缓冲系统中, 直到流规则到达后被删除。缓存系统有效地避免了同一个流触发的重复 Packet-In 消息在控制通道中传输。

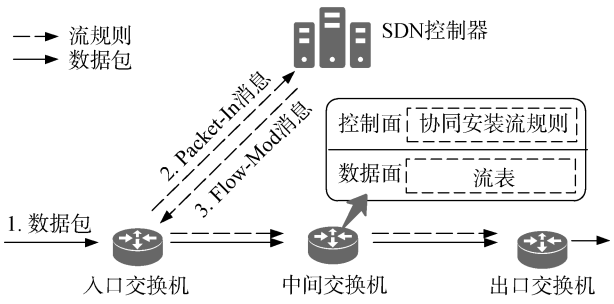


图 22 FastLane 流规则安装过程

Figure 22 Workflow of FastLane rule installation

事实上, 流统计信息采集过程要比流规则安装过程消耗更多的控制通道带宽[92]。SDN 控制器通过频繁地采集交换机中的统计信息(如数据包个数、字

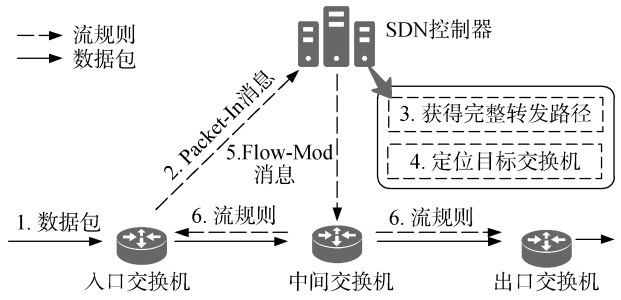


图 23 EFastLane 流规则安装过程

Figure 23 Workflow of EFastLane rule installation

节数和延迟等)及时准确地获得全网流量信息, 并基于此信息实现流量工程、Qos 路由、攻击检测等应用[95-99]。OpenFlow 声明了两种流统计信息采集方法: 一种是 Push 机制, 即交换机主动地向控制器报告已经记录的流量统计信息, 如 FlowSense[21]使用了交换机主动发送的 Packet-In 和 Flow-Remove 消息来统计一个流规则整个生命周期内所记录的统计信息, 然而一些因素限制了 Push 机制在实践中的应用, 例如 Push 机制需要对交换机硬件和软件有一些额外的要求(检测条件满足时触发报告过程); 另一种是 Pull 机制, 即控制器向交换机发送一个 Read-State 请求消息, 然后交换机回复包含流统计信息的响应消息。由于 Pull 机制对交换机的硬件和软件不需要额外的要求, 因此它在 SDN 应用程序中得到了广泛的使用。目前 Pull 采集机制包含两种采集模式: 基于数据流的采集模式和基于交换机的采集模式。OpenNetMon[100]使用一种自适应机制从交换机中获取特定流的统计信息, 当样本之间的差异较大时统计信息的查询速度会增加, 而统计信息趋于稳定时其查询速度会下降。OpenTM[101]定期从交换机中采集每个活动流的统计数据, 并以此估计当前网络的流量矩阵。FlowCover[102] 和 CeMon[103]提出了基于交换机的低成本监控方案以支持各种网络管理任务。Xu 等人[104]证明了基于数据流的采集模式和基于交换机的采集模式会导致控制信道带宽的巨大开销和交换机上较长的处理时延, 同时提出了一种基于通配符的新型统计信息采集机制。通过使用基于通配符的采集方式, 控制器可以一次性获得多个流的统计信息。相比基于数据流的采集模式, 这种方式减少了请求次数; 相比基于交换机的方式, 控制器可以只获得感兴趣的部分流统计信息, 而不是交换机中全部流规则的统计信息。此外, Xu 等人[104]通过仿真实验证明了基于通配符的采集机制比现有统计信息采集机制减少了超过 40%的控制通道带宽消耗。

5.3 数据平面

数据平面是由 SDN 交换机组成, 本小节将分别介绍目前 SDN 交换机的计算资源保护机制和存储资源保护机制。

5.3.1 计算资源保护机制

来自控制平面的恶意控制消息会极大地消耗交换机的计算资源。恶意控制消息的产生根源可能来自于攻击流量、恶意 SDN 应用程序以及非法控制平面。因此现有防御机制主要是从以上三个方面出发, 减少甚至规避恶意消息进入交换机。Avant-Guard[31]和 SDNShield[75]等机制通过在网络入口处部署过滤模块禁止攻击流量进入 SDN 网络, 从而避免恶意流规则的产生。NICE 检测模型[89]对 SDN 应用程序进行漏洞测试与验证, 避免恶意应用部署到控制器中。ProvSDN[15]利用数据源技术追踪信息流, 进而检测未授权应用程序下发流规则的行为。ForenGuard[105]利用静态分析的方式识别 SDN 应用程序中的状态变量, 并利用插桩技术来监控这些变量的读写操作, 最后通过对日志进行分析检测出下发恶意控制消息的 SDN 应用程序。除此之外, 攻击者可以伪装成合法的控制平面向数据平面随意发送恶意控制信息, OpenFlow 白皮书[57]建议采用基于 TLS 协议的安全控制通道以实现控制器和交换机之间的身份认证。

5.3.2 存储资源保护机制

SDN 交换机中最重要的存储资源是用来存储转发规则的流表。为了实现线速转发数据包, 当前 SDN 硬件交换机一般采用 TCAM 作为流表来保存转发规则。但是由于 TCAM 的高成本和高能耗, 流表通常只能够安装数千流规则。Kandoi 等人[106]证明了有限的 SDN 流表很容易遭受溢出攻击。为了防止流表溢出, 研究人员提出了多种保护机制[107-126], 如表 7 所示。我们将这些机制分为预防溢出机制(正常网络状态下)和溢出缓解机制(攻击发生时)。

预防溢出机制在流表溢出之前执行, 主要包括规则聚合和最优超时配置。这些解决方案通过优化流表利用率降低在正常情况下流表溢出的发生概率。流规则聚合机制是指使用一条转发规则来匹配多条数据流, 从而减少流表中的规则数量。根据帕累托法则, 网络中的大部分流量是由少数数据流产生, 因此只要保证这一小部分流可以高效转发便能够有效提高整个网络的性能。Xu 等人[107]提出了一种将传统交换机与 SDN 交换机相结合的混合交换机设计。混合交换机只为大象流分配特定的规则, 而老鼠流则采用传统交换机中基于目标地址的默认规则处理。Nguyen 等人[108]同样地只为大象流安装特定规则, 老

表 7 表溢出攻击缓解机制对比

Table 7 Comparison of mechanisms against table overflow attacks

相关工作	关键技术	技术简述	是否需要修改设备	计算负载	其他不足
Xu et al.[107]	流规则聚合	为大象流分配特定规则, 老鼠流规则聚合为默认规则	是	低	丢失老鼠流的可视性
Nguyen et al.[108]	流规则聚合	为重要数据流分配准确规则, 其余规则聚合为默认规则	是	低	丢失老鼠流的可视性
CacheFlow[109]	存储分类使用	TCAM 负责存储重要规则, 其余规则由 SRAM 存储	是	低	增加流时延
Palette[110]	规则聚合与分解	将保存网络策略整体语义的规则分解为诸多小型表, 分发给个个交换机	否	高	算法相对复杂, 不易实现
Kim et al.[111]	最优空闲超时	控制器根据数据流的报文个数、时延等统计信息动态调整规则超时	否	高	难以实现控制器信息采集周期与准确性之间的平衡
Zhang et al.[112]	最优强制超时	控制器将流表建模为一个排队系统, 根据规则的等待时延和处理时延为每一个流规则推导出最优的强制超时	否	高	采用最基本的排队理论, 忽视了规则之间优先级
SmartTime[113]	最优空闲超时	根据当前 TCAM 利用率和数据流再次出现的可能性来决定流规则的超时	否	高	难以实现控制器信息采集周期与准确性之间的平衡
Zhu et al.[114]	最优空闲超时	根据统计特性为流规则分配适当的超时, 在避免流表溢出的前提下最大化超时值	否	高	难以实现控制器信息采集周期与准确性之间的平衡
Liu et al.[115]	最优空闲超时	根据实际网络状况和 TCAM 资源使用情况在不同采样周期为流规则设置不同超时值	否	高	难以实现控制器信息采集周期与准确性之间的平衡
Kim et al.[117]	被动规则剔除	基于 LRU 算法的规则剔除机制	是	高	由于错误剔除可能造成更高的网络负载
Lee et al.[118-120]	被动规则剔除	基于 LRU 算法的规则剔除机制	是	高	由于错误剔除可能造成更高的网络负载
Tian et al.[121]	被动规则剔除	自适应最少频繁规则驱逐算法	是	高	无法充分释放流表资源
FlowMaster[122]	主动规则剔除	利用马尔可夫模型来预测某一流规则是否仍有存在价值	是	高	网络动态性导致准确率不高
Qiao et al.[123]	攻击迁移	将流量由 TCAM 资源紧张的交换机引导至资源充足的交换机	否	低	无法从根本上防御攻击, 甚至造成更恶劣的影响
Yuan et al.[124]	攻击迁移	集成整个 SDN 系统中可用的空闲流表资源缓解对单个交换机的攻击	否	低	无法从根本上防御攻击, 甚至造成更恶劣的影响
Xu et al.[125]	速率限制	基于令牌桶模型限制攻击流量速率	否	低	无法细粒度地识别攻击流量
Kandoi et al.[126]	速率限制	利用 OpenFlow 中的 Meter 表实现限速	否	低	无法细粒度地识别攻击流量

鼠流则由默认规则处理。CacheFlow[109]只在 TCAM 中存储重要流对应的规则, 其余规则使用软件交换机中的 RAM 存储。Kanizo 等人[110]提出了一个分发框架 Palette, 用于将大型 SDN 表分解为小型 SDN 表, 然后在保留 SDN 策略整体语义的同时向网络中的所有交换机分发这些小型表。此外, 通过优化流规则的超时设置来提高流表的利用率也是一种可行的方案。Kim 等人[111]提出了一种动态超时控制算法。控制器首先从交换机中收集多种流量参数, 如数据包数目、时延等, 并预测数据包的到达时间。基于上述信息, 控制算法确定下一个采样周期可能保留在流表中的流规则数量, 然后动态调整每个流规则的超时值, 以便在流表中为新到达的流规则预留空间。Zhang 等人[112]提出了一个自适应超时方法, 通过优化流规则的强制超时来提高流表的利用率。该方法将流表建模为一个排队系统, 根据规则的等待时延和处理时延为每一个流规则推导出最优的强制超时值。Vishnoi 等人[113]提出了 SmartTime 系统, 它使用自适应启发式算法来计算流规则的空闲超时, 有效地利用了 TCAM 流表资源。SmartTime 综合考虑了当前 TCAM 利用率和数据流再次出现在网络中的可能性, 进而更好地决定流规则的空闲超时。Zhu 等人[114]提出了一种智能超时控制机制, 它可以根据不同的统计特性为流规则分配适当的超时值, 并根据当前流表占用情况进行超时反馈, 从而在避免流表溢出的前提下最大化超时值。Liu 等人[115]提出了一种动态自适应超时算法, 使得控制器根据实际网络状况和 TCAM 资源使用情况在不同采样周期为流规则设置不同的超时值。

上述解决方案侧重于在正常网络状况下提高流表的利用率, 但当网络中存在攻击者并且攻击者试图耗尽流表资源时, 这些方案便会失去作用。攻击者

在短时间内伪造大量攻击流, 迫使控制器集中向交换机安装规则。这样一来, 恶意规则将完全占用流表空间, 从而导致合法流规则无法安装。目前针对流表溢出攻击的缓解机制可大致分为三类: 驱逐机制、攻击流量迁移和速率限制。驱逐机制的原理是通过剔除交换机流表中的某些流规则为安装新规则释放空间, 如图 24 所示。驱逐机制的关键是如何确定被剔除的流规则, 从而使得网络负载最小化。常用的剔除算法包括最近最少使用(Least Recent Used, LRU)、先进先出(First-In-First-Out, FIFO)和随机替换(Random Replacement)等[116]。Kim 等人[117]指出 LRU 优于其他剔除算法, 即通过在流表中保留最近使用的流规则可以提高流表命中率。Lee 等人[118-120]提出了基于 LRU 算法的规则剔除机制, 从而保证合法流规则的正常安装。基于对流量重尾特征的观察以及流量规模与流规则生存时间的统计正相关关系, Tian 等人[121]设计了自适应最少频繁驱逐算法, 并通过实验证明了该算法相比于 LRU 算法可提高超过 15%的流规则命中率。Kannan 等人[122]提出了一个推测性的主动剔除机制, 称为 FlowMaster。它利用马尔可夫模型来预测某一流规则在下一周期是否仍具有存在的价值。对于预测未来周期不会出现的流量, FlowMaster 主动剔除对应的流规则。此外, 由于单个交换机没有足够的 TCAM 资源抵御流表溢出攻击, 研究人员提出了攻击流量迁移机制。Qiao 等人[123]设计了一种流表共享机制, 它将在流表资源相对缺乏的交换机中触发的流规则请求引导至其他流表资源相对充足的交换机中, 使之代替资源紧张的交换机安装控制器响应的规则。Yuan 等人[124]提出了一种基于 Qos 感知的缓解策略, 即对等支持策略(Peer Support Strategy, PSS), 该策略集成了整个 SDN 系统中可用的空闲流表资源以缓解对单个交换机的攻击。速率限制是直接缓解流表溢出攻击的另一种机制。例如 Xu 等人[125]提出了一种基于令牌桶模型限制攻击流量速率的缓解机制, 该机制通过控制令牌添加速率和桶容量避免了受害交换机上的流表溢出。使用 OpenFlow 协议中的 Meter 表同样可以实现对流量速率的限制, 如 Kandoi 等人[126]使用 Meter 表限制了交换机处理数据包的速率。

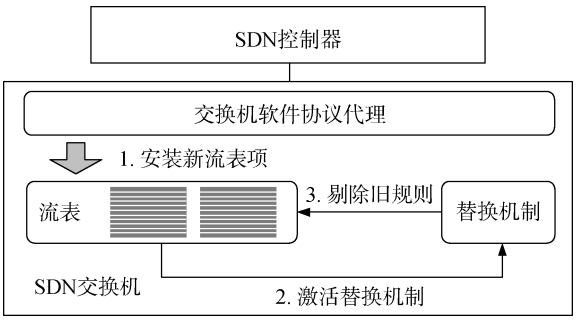


图 24 SDN 交换机中的替换机制

Figure 24 Replacement mechanism in SDN switch

6 讨论和未来方向

本文综述了 SDN 网络中的资源消耗型攻击和防御技术。实际上, 一种类型的攻击可能会造成多种 SDN 资源的消耗, 如 Packet-In 泛洪攻击同时会对交换机计算资源、控制通道带宽资源以及控制器计算资源造成影响。但是, 每一种攻击都是针对一种主要的目标所提出, 例如攻击者针对 SDN 网络中的集中式管理架构(传统网络中控制面与数据面是一对一的, 而 SDN 中控制面与数据面是多对一的)提出 Packet-In 泛洪攻击, 该攻击的主要目的是耗尽控制平面的计算资源, 使整个网络发生单点故障。

从攻击角度来讲, 随着攻击者能力的增强, 资源消耗攻击也愈发隐蔽和精密。例如 Table-miss 增强攻击[51]是一种隐蔽性更强、攻击性能更好的 Packet-In 泛洪攻击, 跳跃式流表溢出攻击[119]是一种新型的流表溢出攻击, 以及 CrossPath 攻击[55]利用共享资源的脆弱性对控制通道的带宽资源进行隐式消耗等。新型攻击的产生使得攻击者可以通过更小的攻击成本、更高的隐蔽性获得更好的攻击效果。

从防御角度来讲, 现有针对资源消耗型攻击的防御机制存在着诸多不足。例如对于 Packet-In 泛洪攻击, 基于熵的机制大多是检测机制而没有实现攻击缓解的功能; 机器学习机制对于数据依赖型强, 而且通常是计算密集型, 不利于方案的在线实施; 攻击迁移机制需要修改 SDN 基础设施或者添加额外的设备, 影响方案的通用性和可部署性。针对数据面的流表溢出攻击, 规则汇聚机制影响了 SDN 控制器对流量的细粒度控制; 驱逐机制带来了较大的网络开销; 攻击流量转移机制无法从根本上解决攻击, 甚至会加大攻击的负面影响。此外, 针对流表溢出攻击, 现有机制主要是以被动防御为主, 而且攻击模型较为简单, 即少量攻击者使某一交换机流表持续发生溢出。优化控制消息的传输机制虽然可以减少控制消息对控制通道带宽资源的占用, 但当攻击流量规模很大时仍无法避免通道阻塞; 控制逻辑下移机制可以减少控制面和数据面的通信, 但同时会减弱控制平面对网络流的可见性。由于资源消耗型攻击影响的广泛性, 我们建议设计一个更加系统的防御方案, 它首先根据攻击影响定位出攻击发生的根

本原因(如 Wang 等人[99]利用回溯技术设计了ForenGuard 系统, 实现了根据数据包转发错误检测出恶意SDN应用程序的功能), 然后根据攻击产生原因调用专门的缓解机制。例如攻击产生的原因是恶意流量的注入, 防御方案则在网络入口处过滤掉恶意流量; 如果攻击产生原因是恶意应用的部署, 防御方案则直接在控制器上卸载该应用。除此之外, 通用性强、可部署性高以及轻量级的防御技术也是未来的研究方向。

7 结论

本文对SDN网络中的资源消耗型攻击和防御机制进行了综述, 介绍了 SDN 网络中易受攻击的关键资源以及多种典型的资源消耗型攻击, 并利用实验平台证明了这些攻击的可行性和有效性。同时, 本文分类整理了现有防御机制的技术路线、实现方法及不足。随着攻击者能力的增强, 资源消耗型攻击愈发隐蔽和复杂, 有效、通用以及低部署成本的防御技术仍然是未来研究的重点。

致 谢 在此向对本文工作提出指导的各位同门表示感谢, 以及对提出建议的评审专家表示感谢。

参考文献

[1] D. Kreutz, M.V.R. Fernanod, P.E. Verissimo, et al. Software-Defined Networking：A Comprehensive Survey[J]. *The IEEE*, 2015, 103(1): 14-76.

[2] N. McKeown, T. Anderson, H. Balakrishnan, et al. OpenFlow: Enabling Innovation in Campus Networks[C]. *ACM SIGCOMM Computer Communication Review*, 2008: 69-74.

[3] Hong S, Baykov R, Xu L, et al. Towards SDN-Defined Programmable BYOD (Bring your Own Device) Security[C]. 2016 Network and Distributed System Security Symposium, 2016: 256-263.

[4] Zeng Y, Guo S T, Liu G Y. Comprehensive Link Sharing Avoidance and Switch Aggregation for Software-defined Data Center Networks[J]. *Future Generation Computer Systems*, 2019, 91: 25-36.

[5] H. Wang, A. Srivastava, L. Xu, et al. Bring your own controller: Enabling tenant-defined SDN apps in IaaS clouds[C]. *IEEE Conference on Computer Communications (INFOCOM'17)*, 2017: 1-9.

[6] Sahay R, Meng W Z, Estay D A S, et al. CyberShip-IoT: A Dynamic and Adaptive SDN-based Security Policy Enforcement Framework for Ships[J]. *Future Generation Computer Systems*, 2019, 100: 736-750.

[7] S. Jain, A. Kumar, S. Mandal, et al. B4: Experience with a globally-deployed software defined WAN[C]. *ACM SIGCOMM Computer Communication Review*, 2013: 3-14.

[8] NSX Virtualization Platform, VMware, https://www.vmware.com/products/nsx.html, 2019.

[9] C.Y. Hong, S. Kandula, R. Mahajan, et al. Achieving high utilization with software-driven WAN[C]. *ACM SIGCOMM Computer Communication Review*, 2013: 15-26.

[10] Y. Zhu, M.H. Ammar. Algorithms for assigning substrate network resources to virtual network components[C]. *IEEE Conference on Computer Communications (INFOCOM'06)*, 2006: 1-12.

[11] S. Perrin, S. Hubbard. Practical Implementation of SDN & NFV in the WAN[OL]. White paper. Heavy Reading, 2013.

[12] Yoon C, Lee S, Kang H, et al. Flow Wars: Systemizing the Attack Surface and Defenses in Software-Defined Networks[J]. *ACM Transactions on Networking*, 2017, 25(6): 3514-3530.

[13] Open vSwitch. Linux Foundation, http://openvswitch.org/, Sept, 2019.

[14] Ryu component-based software defined networking framework. OSRG, http://osrg.github.io/ryu/, Sept, 2019.

[15] B.E. Ujcich, S. Jero, A. Edmundson, et al. Cross-App Poisoning in Software-Defined Net-working[C]. *ACM Conference on Computer and Com-munications Security (CCS' 18)*, 2018: 648-663.

[16] RFC 6241, Network Configuration Protocol. IETF, http://www.rfc-editor.org/info/rfc6241, Jun, 2011.

[17] RFC 8231, Path Computation Element Communication Protocol (PCEP) Extensions for Stateful PCE. IETF, https://datatracker.ietf.org/doc/rfc8231/, Sept, 2017.

[18] RFC 7047, The Open vSwitch Database Management Protocol. IETF, https://datatracker.ietf.org/doc/rfc7047/, Dec, 2013.

[19] RFC 7752, North-Bound Distribution of Link-State and Traffic Engineering (TE) Information Using BGP. IETF, https://datatracker.ietf.org/doc/rfc7752/, Mar, 2016.

[20] RFC 3920, Extensible Messaging and Presence Protocol. IETF, https://datatracker.ietf.org/doc/rfc3920/, OCT, 2004.

[21] Yu C, Lumezanu C, Zhang Y P, et al. FlowSense: Monitoring Network Utilization with Zero Measurement Cost[J]. *Passive and Active Measurement*, 2013: 31-41.

[22] M. Yu, L. Jose, R. Mao. Software Defined Traffic Measurement with OpenSketch[C]. *USENIX Symposium on Networked Systems Design and Implementation (NDSI'13)*, 2013: 29-42.

[23] A. Gupta, R. Harrison, M. Canini, et al. Sonata: Query-Driven Streaming Network Telemetry[C]. *ACM Special Interest Group on Data Communication,* 2018: 357-371.

[24] C.R. Taylor, D.C. MacFarland, D.R. Smestad, et al. Contextual, Flow-Based Access Control with Scalable Host-based SDN Techniques[C]. *IEEE Conference on Computer Communications (INFOCOM'16)*, 2016: 1-9.

[25] Gao S, Li Z C, Yao Y, et al. Software-Defined Firewall: Enabling Malware Traffic Detection and Programmable Security Control[C]. *The 2018 on Asia Conference on Computer and Communications Security*, 2018: 413-424.

[26] Hu H X, Han W, Ahn G J, et al. FLOWGUARD: Building Robust Firewalls for Software-defined Networks[C]. *The third workshop on Hot topics in software defined networking*, 2014: 97-102.

[27] Giotis K, Argyropoulos C, Androulidakis G, et al. Combining OpenFlow and sFlow for an Effective and Scalable Anomaly Detection and Mitigation Mechanism on SDN Environments[J]. *Computer Networks*, 2014, 62: 122-136.

[28] S.K. Fayaz, Y. Tobioka, V. Sekar, et al. Bohatei: Flexible and Elastic DDoS Defense[C]. *USENIX Security Symposium*, 2015: 817-832.

[29] Zheng J, Li Q, Gu G F, et al. Realtime DDoS Defense Using COTS SDN Switches via Adaptive Correlation Analysis[J]. *IEEE Transactions on Information Forensics and Security*, 2018, 13(7): 1838-1853.

[30] S. Shin, P. Porras, V. Yegneswaran, et al. FRESCO: Modular Composable Security Services for Software-Defined Networks[C]. *Annual Network & Distributed System Security Symposium (NDSS'13)*, 2013:265-274.

[31] S. Shin, V. Yegneswaran, P.A. Porras, and G. Gu, "AVANT-GUARD：Scalable and Vigilant Switch Flow Management in Software-Defined Networks," in *Proc. ACM conference on Computer and Communications security (CCS'13)*, pp. 413-424, 2013.

[32] Shin S, Yegneswaran V, Porras P, et al. AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-defined Networks[C]. *The 2013 ACM SIGSAC conference on Computer & communications security*, 2013: 413-424.

[33] H. Wang, L. Xu, G. Gu. Floodguard: A dos attack prevention extension in software-defined networks[C]. *Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'15)*, 2015: 239-250.

[34] S. Gao, Z. Peng, B. Xiao, et al. FloodDefender: Protecting data and control plane resources under SDN-aimed DoS attacks[C]. *IEEE Conference on Computer Communications (INFOCOM'17)*, 2017: 1-9.

[35] Ambrosin M, Conti M, de Gaspari F, et al. LineSwitch: Tackling Control Plane Saturation Attacks in Software-Defined Networking[J]. *ACM Transactions on Networking*, 2017, 25(2): 1206-1219.

[36] Porras P, Shin S, Yegneswaran V, et al. A Security Enforcement Kernel for OpenFlow Networks[C]. *The first workshop on Hot topics in software defined networks*, 2012: 121-126.

[37] Porras P, Cheung S, Fong M, et al. Securing the Software Defined Network Control Layer[C]. *2015 Network and Distributed System Security Symposium*, 2015: 102-110.

[38] S. Shin, Y. Song, T. Lee, et al. Rosemary: A Robust, Secure, and High-performance Network Operating System[C]. *ACM conference on Computer and communications security (CCS'14)*, 2014: 78-89.

[39] T. Koponen, M. Casado, N. Gude, et al. Onix: A distributed control platform for large-scale production networks[C]. *Usenix Conference on Operating Systems Design and Implementation*, 2010: 1-6.

[40] N. Katta, H. Zhang, M. Freedman, et al. Ravana: Controller fault-tolerance in software-defined networking[C]. *ACM Special Interest Group on Data Communication*, 2015: 4-16.

[41] Lee S, Yoon C, Lee C, et al. DELTA: A Security Assessment Framework for Software-Defined Networks[C]. *2017 Network and Distributed System Security Symposium*, 2017: 123-134.

[42] L. Xu, J. Huang, S. Hong, et al. Attacking the Brain: Races in the SDN Control Plane[C]. *USENIX Security Symposium*, 2017: 451-468.

[43] Al-Shaer E, Al-Haj S. FlowChecker: Configuration Analysis and Verification of Federated Openflow Infrastructures[C]. *The 3rd ACM workshop on Assurable and usable security configuration*, 2010: 37-44.

[44] S. Son, S. Shin, V. Yegneswaran, et al. Model checking invariant security properties in OpenFlow[C]. *IEEE international conference on communications (ICC'13)*, 2013: 1974-1979.

[45] C. Prakash, J. Lee, Y. Turner, et al. PGA: Using Graphs to Express and Automatically Reconcile Network Policies[C]. *ACM SIGCOMM Computer Communication Review*, 2015: 29-42.

[46] Hong S, Xu L, Wang H P, et al. Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures[C]. *2015 Network and Distributed System Security Symposium*, 2015: 365-378.

[47] R. Skowyra, L. Xu, G. Gu, et al. Effective Topology Tampering Attacks and Defenses in Software-Defined Networks[C]. *Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'18)*, 2018: 374-385.

[48] Dhawan M, Poddar R, Mahajan K, et al. SPHINX: Detecting Security Attacks in Software-Defined Networks[C]. *2015 Network and Distributed System Security Symposium*, 2015: 568-574.

[49] Shaghaghi A, Kaafar M A, Jha S. WedgeTail: An Intrusion Prevention System for the Data Plane of Software Defined Networks[C]. *The 2017 ACM on Asia Conference on Computer and Communications Security*, 2017: 849-861.

[50] Li Q, Zou X Y, Huang Q, et al. Dynamic Packet Forwarding Verification in SDN[J]. *IEEE Transactions on Dependable and Secure Computing*, 2019, 16(6): 915-929.

[51] B.E. Ujcich, U. Thakore, W.H. Sanders. ATTAIN: An Attack In-

jection Framework for Software-Defined Networking[C]. *Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'17)*, 2017: 567-578.

[52] Jero S, Bu X Y, Nita-Rotaru C, et al. BEADS: Automated Attack Discovery in OpenFlow-Based SDN Systems[M]. Research in Attacks, Intrusions, and Defenses. Cham: Springer International Publishing, 2017: 311-333.

[53] S. Jero, W. Koch, R. Skowyra, et al. Identifier Binding Attacks and Defenses in Software-Defined Networks[C]. *USENIX Security Symposium*, 2017: 415-432.

[54] Shin S, Gu G F. Attacking Software-defined Networks: A First Feasibility Study[C]. *The second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013: 165-166.

[55] Sonchack J, Dubey A, Aviv A J, et al. Timing-based Reconnaissance and Defense in Software-defined Networks[C]. *The 32nd Annual Conference on Computer Security Applications*, 2016: 89-100.

[56] Cui H, Karame G O, Klaedtke F, et al. On the Fingerprinting of Software-Defined Networks[J]. *IEEE Transactions on Information Forensics and Security*, 2016, 11(10): 2160-2173.

[57] Zhang M H, Li G Y, Xu L, et al. Control Plane Reflection Attacks in SDNS: New Attacks and Countermeasures[J]. *Research in Attacks, Intrusions, and Defenses*, 2018: 161-183.

[58] OpenFlow switch specification 1.5.1. Open Networking Foundation, https://www.opennetworking.org/software-defined-standards/specifications/, Apr, 2015.

[59] Hpe sdn app store. Hewlett-Packard Enterprise, https://community.arubanetworks.com/t5/SDN-Apps/ct-p/SDN-Apps, Apr, 2015.

[60] A Switch Table Vulnerability in the Open Floodlight SDN Controller. http://dovernetworks.com/wp-content/uploads/2014/03/OpenFloodlight-03052014.pdf, Apr, 2015.

[61] J. Cao, Q. Li, R. Xie, et al. The CrossPath Attack: Disrupting the SDN Control Channel via Shared Links[C]. *USENIX Security Symposium*, 2019: 19-36.

[62] A.R. Curtis, J.C. Mogul, J. Tourrilhes, et al. DevoFlow: scaling flow management for high-performance networks[C]. *ACM SIGCOMM Computer Communication Review*, 2011: 254-265.

[63] X. Jin, H.H. Liu, R. Gandhi, et al. Dynamic scheduling of network updates[C]. *ACM SIGCOMM Computer Communication Review*, 2014: 539-550.

[64] Pica8: Flow scalability per broadcom chipset. Pica8, https://docs.pica8.com/display/picos2102cg/Flow+Scalability+per+Broadcom+Chipset, Apr, 2015.

[65] P. Bosshart, D.P. Daly, G.Gibb, et al. P4: programming protocol-independent packet processors[C]. *ACM SIGCOMM Computer Communication Review*, 2014: 87-95.

[66] R. Wang, Z.P. Jia, L. Ju. An Entropy-Based Distributed DDoS Detection Mechanism in Software-Defined Networking[C]. *IEEE Trustcom/BigDataSE/ISPA*, 2015: 310-317.

[67] X. Huang, X. Du, B. Song. An Effective DDoS Defense Scheme for SDN[C]. *IEEE International Conference on Communications (ICC'17)*, 2017: 1-6.

[68] L. Dridi, M.F. Zhani. SDN-Guard: DoS attacks mitigation in SDN networks[C]. *IEEE International Conference on Cloud Networking (CloudNet'16)*, 2016: 212-217.

[69] Cui J, Wang M J, Luo Y L, et al. DDoS Detection and Defense Mechanism Based on Cognitive-inspired Computing in SDN[J]. *Future Generation Computer Systems*, 2019, 97: 275-283.

[70] Kalkan K, Altay L, Gur G, et al. JESS: Joint Entropy-Based DDoS Defense Scheme in SDN[J]. *IEEE Journal on Selected Areas in Communications*, 2018, 36(10): 2358-2372.

[71] Deng S H, Gao X, Lu Z B, et al. DoS Vulnerabilities and Mitigation Strategies in Software-defined Networks[J]. *Journal of Network and Computer Applications*, 2019, 125: 209-219.

[72] R. Braga, E. Mota, A. Passito. Lightweight DDoS flooding attack detection using NOXOpenFlow[C]. *IEEE Conference on Local Computer Networks*, 2010: 408-415.

[73] P. Dong, X. Du, H. Zhang, et al. A Detection Method for a Novel DDoS Attack against SDN Controllers by Vast New Low-Traffic Flows[C]. *IEEE International Conference on Communications (ICC'16)*, 2016: 1-6.

[74] Wang T, Chen H C. SGuard: A Lightweight SDN Safe-guard Architecture for DoS Attacks[J]. *China Communications*, 2017, 14(6): 113-125.

[75] A. C. Lapolli, J. A. Marques, L. P. Gaspary. Offloading Real-time DDoS Attack Detection to Programmable Data Planes[C]. *IFIP/IEEE Symposium on Integrated Network and Service Management (IM'19)*, 2019: 19-27.

[76] K.Y. Chen, A.R. Junuthula, I.K. Siddhrau, et al. SDNShield: Towards more comprehensive defense against DDoS attacks on SDN control plane[C]. *IEEE Conference on Communications and Network Security (CNS'16)*, 2016: 28-36.

[77] Deng S H, Gao X, Lu Z B, et al. Packet Injection Attack and Its Defense in Software-Defined Networks[J]. *IEEE Transactions on Information Forensics and Security*, 2018, 13(3): 695-705.

[78] Y. Afek, A.B. Barr, L. Shafir. Network Anti-Spoofing with SDN Data plane[C]. *IEEE Conference on Computer Communications (INFOCOM'17)*, 2017: 1-9.

[79] Cao J M, Bi J, Zhou Y, et al. CoFilter: A High-Performance Switch-Assisted Stateful Packet Filter[C]. Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos, 2018: 9-11.

[80] A. Tootoonchian, Y. Ganjali. Hyperflow: A distributed control plane for openflow[C]. *Internet network management conference on Research on enterprise networking*, 2010: 3-3.

[81] S.H. Yeganeh, and Y. Ganjali, “Kandoo: a framework for efficient and scalable offloading of control applications,” in *Proc. the first ACM workshop on Hot topics in software defined networking*, pp. 19-24, 2012.

[82] Hassas Yeganeh S, Ganjali Y. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications[C]. *The first workshop on Hot topics in software defined networks*, 2012: 19-24.

[83] Brocade Vyatta Controller. Brocade, http://www.brocade.com/content/dam/common/documents/content-types/datasheet/brocade-vyatta-controller-ds.pdf, Sep, 2014.

[84] ONOS Distributed Primitives. Open Networking Foundation, https://wiki.onosproject.org/display/ONOS/Distributed+Primitives, Apr, 2015.

[85] A. Panda, W. Zheng, X. Hu, et al. SCL: simplifying distributed SDN control planes[C]. *USENIX Symposium on Networked Systems Design and Im-plementation (NDSI’17)*, 2017: 329-345.

[86] OpenDaylight source code. Linux Foundation, https://www.opendaylight.org/technical-community/getting-started-for-developers/downloads-and-documentation, Apr, 2015.

[87] Open SDN Controller. Cisco, http://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/open-sdn-controller/datasheet-c78-733458.pdf, Aug, 2015.

[88] Tungsten Fabric. Linux Foundation, https://tungstenfabric.github.io/website/Tungsten-Fabric-Architecture.html, 2018.

[89] Contrail Controller. Juniper, http://www.juniper.net/assets/us/en/local/pdf/whitepapers/2000535-en.pdf, Sept, 2015.

[90] T. Ball, N. Bjorner, A. Gember, et al. VeriCon: towards verifying controller programs in software-defined networks[C]. *ACM Sigplan Notices*, 2014: 282-293.

[91] M. Canini, D. Venzano, P. Peresini, et al. A NICE way to test openflow applications[C]. *USENIX Symposium on Networked Systems Design and Implementation (NDSI’12)*, 2012: 127-140.

[92] C. Lee, C. Yoon, S. Shin, et al. INDAGO：A New Framework for Detecting Malicious SDN Applications[C]. *IEEE International Conference on Network Protocols (ICNP’18)*, 2018: 220-230.

[93] Yu M L, Rexford J, Freedman M J, et al. Scalable Flow-based Networking with DIFANE[J]. *ACM SIGCOMM Computer Communication Review*, 2010, 40(4): 351-362.

[94] K. Bu. Gotta Tell You Switches Only Once: Toward Bandwidth-Efficient Flow Setup for SDN[C]. *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2015: 492-497.

[95] J. Xu, L. Wang, C. Song, et al. EFastLane: Toward Bandwidth-Efficient Flow Setup in Software-Defined Networking[C]. *IEEE Wireless Communications and Networking Conference (WCNC’19)*, 2017: 1-7.

[96] F. Li, J. Cao, X. Wang, et al. Adopting SDN Switch Buffer：Benefits Analysis and Mechanism Design[C]. *International Conference on Distributed Computing Systems (ICDCS’17)*, 2017: 2171-2176.

[97] H. Xu, X. Li, L. Huang, et al. High-throughput anycast routing and congestion-free reconfiguration for sdns[C]. *IEEE/ACM International Symposium on Quality of Service (IWQoS’16)*, 2016: 1-6.

[98] Wang B, Zheng Y, Lou W J, et al. DDoS Attack Protection in the Era of Cloud Computing and Software-Defined Networking[J]. *Computer Networks*, 2015, 81: 308-319.

[99] Yan Q, Yu F R, Gong Q X, et al. Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, some Research Issues, and Challenges[J]. *IEEE Communications Surveys & Tutorials*, 2016, 18(1): 602-622.

[100] Zheng J, Li Q, Gu G F, et al. Realtime DDoS Defense Using COTS SDN Switches via Adaptive Correlation Analysis[J]. *IEEE Transactions on Information Forensics and Security*, 2018, 13(7): 1838-1853.

[101] Sun P H, Hu Y X, Lan J L, et al. TIDE: Time-relevant Deep Reinforcement Learning for Routing Optimization[J]. *Future Generation Computer Systems*, 2019, 99: 401-409.

[102] [100] N.L.M.V. Adrichem, C. Doerr, F.A. Kuipers. OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks[C]. *IEEE Network Operations and Management Symposium (NOMS’14)*, 2014: 1-8.

[103] Tootoonchian A, Ghobadi M, Ganjali Y. OpenTM: Traffic Matrix Estimator for OpenFlow Networks[M]. Passive and Active Measurement. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010: 201-210.

[104] Z. Su, T. Wang, Y. Xia, et al. Flowcover: Low-cost flow monitoring scheme in software defined networks[C]. *IEEE Global Communications Conference (Globalcom’14)*, 2014: 1956-1961.

[105] Su Z Y, Wang T, Xia Y, et al. CeMon: A Cost-effective Flow Monitoring System in Software Defined Networks[J]. *Computer Networks*, 2015, 92: 101-115.

[106] H. Xu, Z. Yu, C. Qian, et al. Minimizing Flow Statistics Collection Cost of SDN Using Wildcard Requests[C]. *IEEE Conference on Computer Communications (INFOCOM’17)*, 2017: 1-9.

[107] Wang H P, Yang G L, Chinprutthiwong P, et al. Towards Fine-grained Network Security Forensics and Diagnosis in the SDN Era[C]. *The 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018: 3-16.

[108] R. Kandoi, M. Antikainen. Denial-of-service attacks in OpenFlow SDN networks[C]. *IFIP/IEEE International Symposium on Integrated Network Management (IM’15)*, 2015: 1322-1326.

[109] H. Xu, H. Huang, S. Chen, et al. Scalable software-defined networking through hybrid switching[C]. *IEEE Conference on Com-*

puter Communications (INFOCOM'17), 2017: 1-9.

[110] Nguyen X N, Saucez D, Barakat C, et al. Optimizing Rules Placement in OpenFlow Networks: Trading Routing for Better Efficiency[C]. *The third workshop on Hot topics in software defined networking*, 2014: 127-132.

[111] Katta N, Alipourfard O, Rexford J, et al. CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks[C]. *The Symposium on SDN Research*, 2016: 6-18.

[112] Y. Kanizo, D. Hay, I. Keslassy. Palette: Distributing tables in software-defined networks[C]. *IEEE Conference on Computer Communications (INFOCOM'13)*, 2013: 545-549.

[113] Kim T, Lee K, Lee J, et al. A Dynamic Timeout Control Algorithm in Software Defined Networks[J]. *International Journal of Future Computer and Communication*, 2014, 3(5): 331-336.

[114] L. Zhang, R. Lin, S. Xu, et al. AHTM：Achieving Efficient Flow Table Utilization in Software Defined Networks[C]. *IEEE Global Communications Conference (Globalcom'14)*, 2014 1897-1902.

[115] Vishnoi A, Poddar R, Mann V, et al. Effective Switch Memory Management in OpenFlow Networks[C]. *The 8th ACM International Conference on Distributed Event-Based Systems*, 2014: 177-188.

[116] H. Zhu, H. Fan, X. Luo, et al. Intelligent Timeout Master: Dynamic Timeout for SDN-based Data Centers[C]. *IFIP/IEEE International Symposium on Integrated Network Management (IM'15)*, 2015: 734-737.

[117] Y. Liu, B. Tang, D. Yuan, et al. A Dynamic Adaptive Timeout Approach for SDN Switch[C]. *IEEE International Conference on Computer and Communications (ICCC'16)*, 2016: 2577-2582.

[118] Nguyen X N, Saucez D, Barakat C, et al. Rules Placement Problem in OpenFlow Networks: A Survey[J]. *IEEE Communications Surveys & Tutorials*, 2016, 18(2): 1273-1286.

[119] E. Kim, S. Lee, Y. Choi, et al. A flow entry management scheme for reducing controller overhead[C]. *International Conference on Advanced Communication Technology*, 2014: 754-757.

[120] Kim E D, Choi Y, Lee S I, et al. Enhanced Flow Table Management Scheme with an LRU-Based Caching Algorithm for SDN[J]. *IEEE Access*, 2017, 5: 25555-25564.

[121] E. Kim, Y. Choi, S. Lee, et al. Flow table management scheme applying an LRU caching algorithm[C]. *International Conference on Information and Communication Technology Convergence (ICTC'14)*, 2014: 335-340.

[122] B. Lee, R. Kanagavelu, K.M. Aung. An efficient flow cache algorithm with improved fairness in Software-Defined Data Center Networks[C]. *IEEE International Conference on Cloud Networking (CloudNet'13)*, 2013: 18-24.

[123] T. Pan, X. Guo, C. Zhang, et al. ALFE：A Replacement Policy to Cache Elephant Flows in the Presence of Mice Flooding[C]. *IEEE International Conference on Communications (ICC'12)*, 2012: 2961-2965.

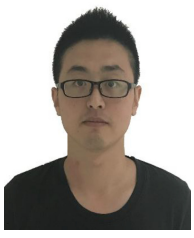
[124] Kannan K, Banerjee S. FlowMaster: Early Eviction of Dead Flow on SDN Switches[J]. *Distributed Computing and Networking*, 2014: 484-498.

[125] S. Qiao, C. Hu, X. Guan, et al. Taming the Flow Table Overflow in OpenFlow Switch[C]. *ACM SIGCOMM Conference*, 2016: 591-592.

[126] Yuan B, Zou D Q, Yu S, et al. Defending Against Flow Table Overloading Attack in Software-Defined Networks[J]. *IEEE Transactions on Services Computing*, 2019, 12(2): 231-246.

[127] Xu T, Gao D Y, Dong P, et al. Mitigating the Table-Overflow Attack in Software-Defined Networking[J]. *IEEE Transactions on Network and Service Management*, 2017, 14(4): 1086-1097.

[128] Y. Qian, W. You, K. Qiao. OpenFlow Flow Table Overflow Attacks and Countermeasures[C]. European Conference on Networks and Communications (EuCNC'16), 2016: 205-209.



徐建峰 于 2016 年在中国海洋大学计算机科学与技术专业获得学士学位。现在中国科学院信息工程研究所攻读博士学位。研究领域为软件定义网络、网络安全。研究兴趣包括: 软件定义网络脆弱性挖掘、网络测量。Email: xujianfeng@iie.ac.cn

徐震 于 2005 年在中国科学院软件所计算机科学与技术专业获得博士学位。现任中国科学院信息工程研究所第五研究室主任, 正高级工程师, 博士生导师。研究领域为云计算安全、可信计算、移动互联网安全、网络与系统安全等。Email: xuzhen@iie.ac.cn

王利明 于 2007 年中国科学院软件所计算机科学与技术专业获得博士学位。现任中国科学院信息工程研究所第五研究室正高级工程师, 博士生导师。研究领域为云计算与网络安全、移动通信系统安全、大数据安全分析、关键基础设施安全。Email: wangliming@iie.ac.cn