一种基于OpenFlow的SDN访问控制策略实时冲突检测与解决方法

王 鹃[1),2)] 王 江[1)] 焦虹阳[1)] 王 勇[1)] 陈诗雅[1)] 刘世辉[1)] 胡宏新[3)]

[1)](武汉大学计算机学院 武汉 430072)

[2)](教育部空天信息安全与可信计算重点实验室 武汉 430072)

[3)](克莱姆森大学 克莱姆森 29634 美国)

摘 要 软件定义网络SDN(Software-Defined Networking)是由美国斯坦福大学Clean Slate研究组提出的一种新型网络创新架构,可通过软件编程的形式定义和控制网络,其控制平面和转发平面分离及开放性可编程的特点,为新型互联网体系结构研究提供了新的实验途径,也极大地推动了下一代互联网的发展.OpenFlow是SDN的主要协议,定义了SDN控制器与交换机之间的通信标准.目前,很多基于OpenFlow的SDN设备已经在实际中得到了部署.但是,基于OpenFlow的SDN却面临很多安全挑战.其中一个重要的挑战是如何建立一个安全可靠的SDN防火墙应用.由于OpenFlow协议的无状态性,现有的SDN防火墙可以被通过改写交换机中的流表项轻松绕过.针对这一安全威胁,作者提出了基于Flowpath的实时动态策略冲突检测与解决方法.通过获取实时的SDN网络状态,能够准确地检测防火墙策略的直接和间接违反,并且一旦发现冲突,可以基于Flowpath进行自动化和细粒度的冲突解决.最后,作者在开源控制器Floodlight上实现了一个安全增强的防火墙应用FlowVerifier,并基于Mininet对FlowVerifier的性能进行了评估.结果表明FlowVerifier能够检测和自动化地解决SDN网络中由于流表改写而引入的策略冲突及其带来的安全威胁.

关键词 软件定义网络;OpenFlow;策略;冲突检测与解决;访问控制

中图法分类号 TP309 **DOI号** 10.3724/SP.J.1016.2015.00872

A Method of OpenFlow-Based Real-Time Conflict Detection and Resolution for SDN Access Control Policies

WANG Juan[1),2)] WANG Jiang[1)] JIAO Hong-Yang[1)] WANG Yong[1)]

CHEN Shi-Ya[1)] LIU Shi-Hui[1)] HU Hong-Xin[3)]

[1)](*School of Computer, Wuhan University, Wuhan* 430072)

[2)](*Key Laboratory of Aerospace Information Security and Trust Computing, Ministry of Education, Wuhan* 430072)

[3)](*Clemson University, Clemson, USA* 29634)

Abstract Software-Defined Networking (SDN) is an innovational network framework introduced by Clean Slate at Stanford University. It enables programmers to control and define the networks by software programming. Additionally, SDN separates data plane and control plane in the networks, and it provides open API and programmability. All of these features provide a new way for the study of new Internet architecture, and have greatly promoted the development of Internet. OpenFlow is a standard protocol of SDN, which defines the communication protocol between SDN controllers and switches. Nowadays, many SDN devices based on OpenFlow have

收稿日期:2014-05-10;最终修改稿收到日期:2014-11-04.本课题得到国家"九七三"重点基础研究发展规划项目基金(2014CB340600)、国家自然科学基金(61173138,61402342,61003628)资助.**王 鹃**,女,1976年生,博士,副教授,主要研究方向为网络与系统安全、安全协议、身份认证、访问控制.E-mail:jwang@whu.edu.cn.**王 江**,男,1992年生,本科,主要研究方向为软件定义网络安全、网络体系结构、云计算.**焦虹阳**,女,1993年生,本科,主要研究方向为网络安全.**王 勇**,男,1991年生,硕士,主要研究方向为云和SDN安全.**陈诗雅**,女,1993年生,本科,主要研究方向为数据中心安全.**刘世辉**,男,1990年生,硕士,主要研究方向为软件定义网络安全.**胡宏新**,男,1974年生,博士,教授,主要研究领域为软件定义网络安全、社交网络中的安全与隐私、网络与系统安全、安全软件工程、访问控制.

been deployed. However, it is faced with many security challenges and one of the most critical challenges is how to implement a secure and reliable SDN firewall application. Due to the statelessness of OpenFlow protocol, the existing firewall security policy for SDN could be easily bypassed by rewriting the flow entries in the switches. To address such a threat, we present a novel approach for real-time policy conflict detection and resolution based on Flowpath. Our approach can accurately detect and effectively resolve policy conflicts through acquiring the network state of SDN in real time. In addition, we present FlowVerifier architecture and implement the SDN firewall application based on our proposed approach in Floodlight. We also evaluate the performance and effectiveness of FlowVerifier in Mininet. Our evaluation results demonstrate that FlowVerifier can automatically detect and resolve the threats of policy conflicts induced by rewriting flow entries.

Keywords　software-defined networking; OpenFlow; policy; conflict detection and resolution; access control

1　引　言

软件定义网络 SDN(Software-Defined Networking)是由美国斯坦福大学 Clean Slate[①] 研究组提出的一种新型网络创新架构,可通过软件编程的形式定义和控制网络,被认为是网络领域的一场革命. SDN 的本质特点是控制平面和数据平面的分离以及开放可编程性. 通过分离控制平面和数据平面以及开放的通信协议,SDN 打破了传统网络设备的封闭性. 此外,南北向和东西向的开放接口及可编程性,也使得网络管理变得更加简单、动态和灵活.

OpenFlow 是目前 SDN 的主流协议,其定义了 SDN 控制器与交换机之间的通信标准. SDN 控制器通过 OpenFlow 协议[②]对 OpenFlow 交换机中的流表进行控制. 控制器会为特定的工作负载计算最佳路径,从而对交换机的数据转发定义路径. 控制器可以是一个设备,一个虚拟机或是一个物理服务器. 目前基于 OpenFlow 的 SDN 已经在美国斯坦福大学、Internet2、日本的 JGN2plus 等多个科研机构中得到部署,一些网络设备生产商也推出了支持 OpenFlow 的有线和无线交换设备. Google 等公司已经在其云数据中心部署了 SDN. 国内,清华大学、中国科学院、北京邮电大学等以及一些知名企业,如华为、百度、腾讯、阿里等也都开始研究和部署 SDN 实验平台. 根据美国著名市场研究公司 Infonetics 的调查数据,未来几年 SDN 的市场价值将超过 30 亿美元. Gartner 在 2013 年发布的 IT 十大战略性技术趋势报告中也将 SDN 列为未来五年的十大关键技术之一.

随着 SDN 的发展和相关网络设备的应用,基于 OpenFlow 的 SDN 网络也面临很多新的安全挑战,其中一个重要的挑战是如何建立一个安全可靠的防火墙. 由于 OpenFlow 协议几乎是无状态的,当主机或网络设备发送信息时,只有当前流的第一个包会被控制器检测,而接下来的包都会直接通过交换机. 因此,攻击者可以通过改写交换机中的流表项而轻松绕过 SDN 防火墙.

针对上述安全威胁,我们提出了一种基于 Flowpath 的实时动态策略冲突检测与解决方法. 通过获取实时 SDN 网络状态,能够准确地检测防火墙策略的直接和间接违反,并且一旦发现冲突,可以基于 Flowpath 进行自动化和细粒度的冲突解决. 最后,我们在开源控制器 Floodlight[③] 上实现了一个安全增强的防火墙应用 FlowVerifier,并在 Mininet 下对 FlowVerifier 进行了性能评估. 结果表明 FlowVerifier 能够检测和自动化地解决 SDN 网络中由于流表改写而引入的策略冲突及其带来的安全威胁. 论文的主要贡献如下:

(1) 分析了 SDN 中的安全威胁和挑战. 通过两个安全策略冲突的例子描述了目前 OpenFlow 网络中安全策略违反的威胁并分析了引起防火墙安全策略被绕过的的主要原因.

(2) 提出了一种基于 Flowpath 的 SDN 策略动态冲突检测方法. 该方法通过获取实时的 SDN 网络状态,并比较防火墙的安全策略空间和对应的

① Clean Slate. http://cleanslate.stanford.edu/2014,4,21
② OpenFlow. OpenFlow1.1.0Specification. http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf 2014,4,30
③ Floodlight. http://www.projectfloodlight.org/documentation/2014,4,30

Flowpath 空间,对防火墙的安全策略进行动态的冲突检测.

(3)提出自动化的冲突解决机制. 一旦检测到策略冲突,根据不同情况基于 Flowpath 对冲突进行自动化的解决,如可以通过在交换机的入口或出口插入特殊的阻断流表来阻止恶意的信息流,进行冲突解决.

(4)实现与评估. 基于 Floodlight 实现了一个安全增强的 SDN 防火墙应用 FlowVerifier. 同时,对 FlowVerifier 的性能进行了评估. 结果表明 FlowVerifier 能够有效地解决由于策略冲突带来的安全威胁.

2 SDN 网络的安全挑战

SDN 作为一种新型的网络架构,出现时间较短. 作为其核心协议,OpenFlow 规定了控制器和交换机通信的数据标准. 控制器向交换机发送符合 OpenFlow 标准的命令. 一个 OpenFlow 交换机可能会包含多个流表,一个流表可能包含多个流表项. 一条流表项包含包头域、计数器域和操作域. 包头域指出了需要匹配的数据包的信息,包括输入端口,源、目的 MAC 地址,源、目的 IP 地址等;计数器域则表示匹配到此条流表项的数据包的个数或者匹配到此流表项的某个类型的数据包的个数或字节数;操作域则是一个集合,包含转发、丢弃、修改等操作. 当数据包被转发时,它将与交换机中的流表项进行匹配,如存在多个可以被匹配的流表项则仅匹配优先级最高的那个. 当流表中没有可以匹配的流表项时,那么交换机则把数据包以 Message-In 的形式发送给控制器,由控制器通过广播信息进行 MAC 地址学习,找到转发的数据通路. 流表项中的 Set-Field 操作可以对数据包的包头进行重写,从而达到对网络结构的灵活控制和可编程目的. SDN 网络的这种灵活性导致其面临一些新的安全挑战[1-2].

首先,OpenFlow 协议几乎是无状态的. 当一条信息流在交换机中没有找到流表项与之匹配时,控制器只接受该信息流的第一个数据包并对其进行检查. 随后数据包会通过交换机直接转发. 因此,攻击者可以利用流表项的 Set-Field 操作构造一个可以绕过 SDN 防火墙安全策略的恶意流,如图 1 所示. 图 1 的网络拓扑中有 4 台主机、3 台交换机和 1 台控制器. 位于控制器的防火墙应用定义了一条安全规则,阻止主机 A(IP:10.0.1.12)和主机 C(IP:10.0.3.32)进行通信. 假设 A、D 为企业内部网络的主机,B 为外部网络的主机,但通过了企业的认证,C 为外部网络未通过企业认证的主机. 现在主机 A 的恶意使用者可以通过以下 3 个交换机的流表项(如图 1 所示),将企业内部的机密信息发送到外部网络未通过验证的 C 主机. 具体实现过程如下:第 1 台交换机的流表项把源地址为 10.0.1.x 且目的地址为 10.0.2.x 的数据包的源地址改为 10.0.4.x;第 2 台交换机的流表项把源地址为 10.0.4.x 且目的地址为 10.0.2.x 的数据包的目的地址改为 10.0.3.x;最后一台交换机的流表项转发源地址为 10.0.4.x 且目的地址为 10.0.3.x 的数据包. 在这种情况下,主机 A(10.0.4.22)可通过给主机 B(10.0.2.22)发送数据包而绕过防火墙. 该数据包可以绕过防火墙是因为它和防火墙的安全策略不匹配,但是通过上文所述的流表操作这个数据包最终会被发到主机 C

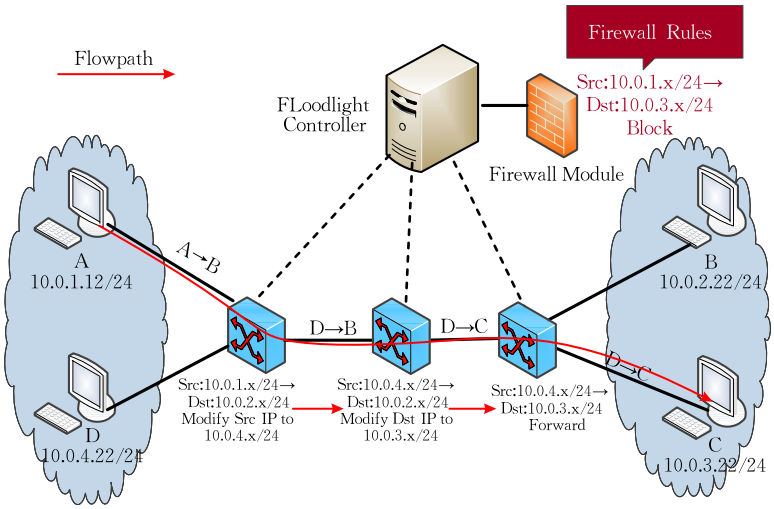

图 1 重写操作导致安全策略被绕过

而不是主机 B. 这样，主机 A 实现了绕过防火墙与主机 C 通信的目的. 通过这个例子，我们可以看到目前基于 OpenFlow 的 SDN 网络存在防火墙被通过改写流表而绕过的安全威胁.

其次，现有的 SDN 防火墙忽略了流表项之间的依赖性，从而使得由于流表项之间的依赖关系而导致防火墙的安全策略违反. 如图 2 所示，假设网络中一个应用程序已经把一条流(Flow 1)的 3 个转发规则安装到交换机. 第 1 台交换机中第 1 条流表项匹配从 10.0.4.x 到 10.0.2.x 的数据包并将其源地址改成 10.0.1.x. 第 2 台交换机中第 1 条流表项匹配从 10.0.1.x 到10.0.2.x 的数据包并将数据包的源地址由 10.0.1.x 改成 10.0.4.x，目的地址由 10.0.2.x 改成 10.0.3.x. 第 3 台交换机中的第 1 条流表项匹配从 10.0.4.x 到 10.0.3.x 的数据包并直接转发. 通过这 3 条流表，类似于图 1，主机 A(IP：10.0.1.12)借助于主机 B(IP：10.0.2.22)将信息发送给了主机 C(IP：10.0.3.22). 此时，假设还有另外一个应用程序把另外一条流(Flow 2)的策略安装到交换机，此策略只包含 3 条转发规则，都是把来自主机 A(IP：10.0.1.12)的数据包转发到主机 B(IP：10.0.2.22). 我们可以进一步假设 Flow 2 中的策略优先级比 Flow 1 的优先级低. 在第 1 台交换机中，数据包转发正常. 当数据包到达第 2 台交换机时，由于两条流表项都是匹配从 10.0.1.x 到 10.0.2.x 的数据包，而由假设可知，无论是来自 Flow 1 还是来自 Flow 2 的数据包，都会按照第 1 条流表项进行处理. 所以来自 Flow 2 的数据包原本应该从主机 A(IP：10.0.1.12)发到主机 B(10.0.2.22)，结果却发到了主机 C(10.0.3.32). 这是我们发现的一种由于流表的优先级关系而导致防火墙规则被违反的情形. 因此，即使所有不同流定义的策略都不违反防火墙安全策略，但是他们之间的依赖关系可能会导致安全策略违反情况的发生.

针对第 1 种防火墙安全策略被绕过的情况，美国德克萨斯农工大学安全通信和计算机系统实验室的研究小组在“A Security Enforcement Kernel for OpenFlow Networks”论文中设计了一种增强安全的 SDN 操作系统内核，其中提出了基于别名集的 SDN 安全策略冲突检测方法. 该方法将通信中源地址和目的地址分别放到两个地址集合中，并将 Set-Field 操作中被修改的地址也加入到上述两个集合，然后将源地址集合和目的地址集合与防火墙的安全规则进行比较从而发现策略冲突. 别名集方法可以找到一些简单的由于修改流表造成的安全策略冲突. 然而，因为它并不基于网络的实时状态，所以在一些复杂的实例中会产生误报. 以图 1 中的网络拓扑为例，防火墙规则为禁止 A 与 C 通信，假设 3 个交换机中的流表项如下：

S1：Match(src＝A，dst＝B)→Set(src＝D，dst＝B)

S2：Match(src＝D，dst＝B)→Set(src＝D，dst＝C)

S3：Match(src＝D，dst＝C)→Set(src＝D，dst＝B)

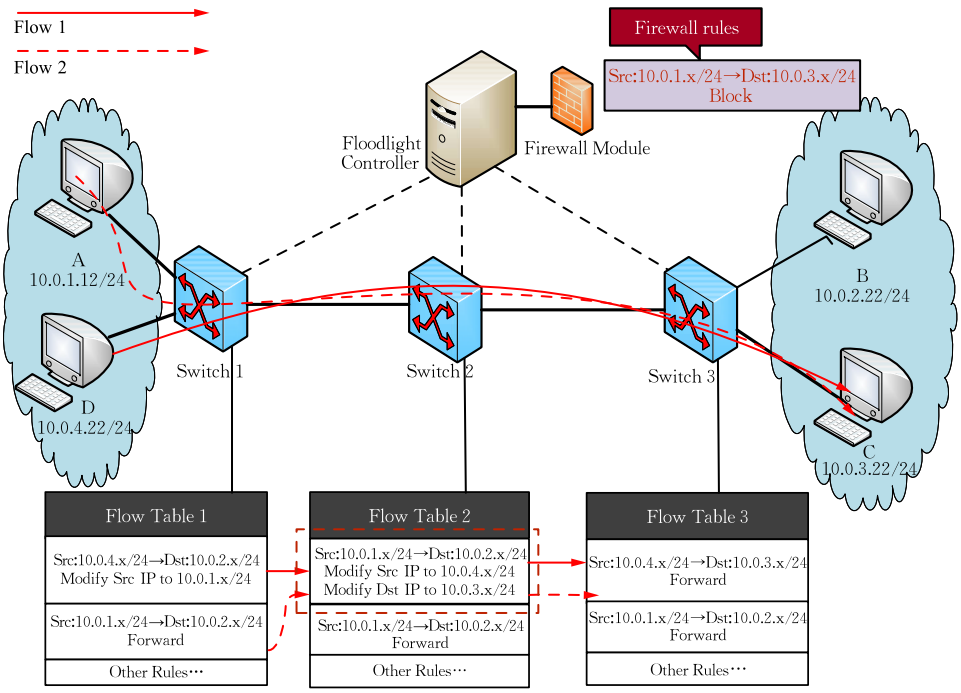

图 2　规则依赖导致安全策略被绕过

S1 交换机匹配源地址是 A 且目的地址是 B 的数据包，然后把它的源地址改为 D；S2 交换机把源地址是 D 且目的地址是 B 的数据包的目的地址改为 C；S3 交换机把源地址是 D 且目的地址是 C 的数据包的目的地址改为 B. 按照别名集的冲突检测算法，会得出(A，D)→(B，C)之间存在冲突，即 A 与 C 之间存在冲突. 但事实上，A 发给 B 的数据包，最终到达了 B，而没有到达 C，即别名集算法发生了误报.

针对上述问题，我们提出了一种基于 Flowpath 的 SDN 安全策略动态检测和解决方法. 通过比较 Shifted Flowpath Space 和防火墙 Deny Authorization Space，对防火墙安全策略进行动态冲突检测和自动化的冲突解决.

3 基于 Flowpath 的 SDN 安全策略动态冲突检测方法

3.1 全网的 Flowpath

Flowpath 指的是 SDN 中每个 Flow 的转发路径，我们所提出的 SDN 安全策略动态冲突检测和解决方法建立在 Flowpath 探测的基础上. Flowpath 的建立基于 Header Space Analysis (HSA)[3-4] 算法. HSA 提供了一个数据包路径计算模型. 数据包的包头可以看做一个 HeaderSpace 空间$\{0,1\}^L$中的一个点，L 为包头最大长度. 网络设备的转发操作可以根据 HeaderSpace 空间的转发规则描述成一个转换函数 T. 这个转换函数 T 可以模拟转发操作，例如接收数据包动作，然后将其转发到相应的输出端口，即

$$T:(h,p)\rightarrow\{(h_1,p_1)(h_2,p_2)\cdots\}.$$

例如，定义 a 和 b 之间的转换函数为

$$R_{a\rightarrow b}=\bigcup_{a\rightarrow b paths}\{T_n(\Gamma(T_{n-1}(\cdots(\Gamma(T_1(h,p)\cdots)))))\},$$

所以，a 和 b 之间的路径是一些转换函数的组合，每个组合的形式如下：

$$\{T_1,T_2,T_3,\cdots,T_n\}.$$

具体实现时，我们将 a 和 b 之间的路径(Flowpath)描述为如下(switch，rule)序列，其中 rule 是交换机的处理规则.

$$(s_1,r_1)\rightarrow\cdots\rightarrow(s_{n-1},r_{n-1})\rightarrow(s_n,r_n).$$

获取 Flowpath 的主要流程如算法 1 所示. 我们获取头节点的头部作为一条 Flowpath 的起点，根据网络的拓扑结构即可得到下一跳(交换机)，然后获取交换机中的流表项的头部信息，将它们转换成二进制向量. 将头结点头部信息中的目的地址和下一跳头部信息中的源地址进行二进制交运算，若交集不为空，则将下一跳加入到 Flowpath 中，这样我们就找到了一条 Flowpath 的一部分. 接下来，通过网络拓扑获得下一跳交换机，同样，比较上一跳头部信息中的目的地址和下一跳的源地址，若存在交集，那么我们又找到了这条 Flowpath 的另外一部分，重复这个过程，直到找到目的节点，一条 Flowpath 就形成了.

算法 1. Flowpath 探测算法.

```
输入：flowtable；topology
输出：flow_path[]
  find_flow_path(flowtable；topology)
  flow_path.path[]←[]
  flow_path.src←first_flow_entry.src；
  switch_linked_list=null；
  WHILE (nexthop，null) DO
    FOR(i=1；i≤nexthop.flowtable.size‖switch==
        null；i++) DO
    switch=get_switch_by_port(topology；
        flow_entry[i].outport)；
    IF (switch!=null) THEN
      flow_path.path.addhop(switch)；
      add_switch linked_list(switch)；
    END IF
      break；
    END FOR
    nexthop=switch_linked_list.next；
  END WHILE
  flow_path.dst←last_flow_entry.dst；
return flow_path[]；
```

3.2 算法复杂度分析

假设 $topology$ 中有 m 个主机，n 个交换机，平均每个交换机中的流表项数目为 k.

根据算法 1 的描述，从 m 个主机中任选一主机开始遍历，根据拓扑结构获取到该主机连接的交换机，并查找交换机中的流表条目与主机、端口进行匹配.

流表对算法复杂度的影响：

(1) 最佳情况：第一次就匹配成功，匹配操作只执行 1 次.

(2) 最坏情况：最后一次匹配成功，匹配操作执行 k 次.

(3) 平均情况：匹配操作执行 $k/2$ 次.

拓扑结构对算法复杂度的影响：

(1) 若 n 个交换机排列成线型网络，则路径的探测算法最多要经过 n 个交换机，最少经过 1 个交换机，平均经过 $n/2$ 个交换机.

(2) 若 n 个交换机排列成网状，经由的交换机

最多小于 n，最少为 1.

(3)若 n 个交换机排列成树状，情况与网状类似.

综上，找出全网络中所有 Flowpath 的时间复杂度为

最坏情况：$m\times k\times n$ 次比较

平均情况：$m\times(k/2)\times(n/2)$次比较

最佳情况：m 次比较

3.3　重写的 Flowpath 空间与 Deny 授权空间

本文中关注的是防火墙安全策略的间接违反，即每一条流表的策略都没有直接违反防火墙的策略，但是通过若干条流表的修改和转发，最终却绕过了防火墙的安全策略. 由于防火墙安全策略的间接违反必须通过修改流表项才能成功，因此我们在进行冲突检测时只需要关注那些存在重写操作的 Flowpath，我们称之为 Shifted Flowpath，所有的 Shifted Flowpath 构成 Shifted Flowpath Space. 由这些 Shifted Flowpath 组成的图称作为 Shifted Flowpath Graph. 此外，防火墙中的 Deny 规则也构成了 Deny 授权空间(Deny Authorization Space). 在检测防护墙策略的间接违反时，我们需要比较 Deny Authorization Space 和 Shifted Flowpath Space.

4　冲突检测与解决

4.1　方案概要

通常情况下，防火墙规则包含 5 个域：源地址、源端口、目的地址、目的端口和协议. Flowpath 的入口头部包含 3 个域：源地址、源端口和协议. Flowpath 的出口头部包含两个域：目的地址和目的端口. 为了检测防火墙规则是否与 OpenFlow 交换机中的流表规则冲突，我们从防火墙规则中提取源地址和目的地址，并且根据该规则计算与之对应的 Shifted Flowpath. 如果存在与防火墙规则源地址和目的地址对应的 Shifted Flowpath，则可以确定防火墙策略和流表策略之间存在冲突. 在检测策略冲突时，我们还需要考虑不同的情况，如添加新的防火墙规则和更新防火墙规则以及添加新的流表项和更新流表项，以便在防火墙规则发生改变时和网络状态发生改变时重新检测防火墙策略空间与流表策略空间的冲突.

一旦安全策略的冲突被检测出来，这些冲突需要自动化地解决，因为在一个较大的网络里，由管理员手工解决冲突是非常困难的. 由于安全策略的违反可能是完全的也可能仅是部分违反，因此需要根据不同的情况进行不同的冲突解决. 对于完全的策略违反，我们可以将这条 Flowpath 直接从网络中移除或者拒绝那些完全违反安全策略的流表项插入交换机. 对于部分安全策略违反，不能够直接移除 Flowpath 或者拒绝流表项的插入，因为它们可能与其他的流存在依赖关系，从而可能会影响其他的 SDN 应用. 针对这个问题，我们可以通过添加更高优先级的 Deny 规则来解决策略的部分违反. 具体在冲突解决时，也需要考虑以下各种不同的情况，从而进行自动化和细粒度的冲突解决.

4.2　添加防火墙规则和更新防火墙规则

向防火墙中添加新的规则可能会导致防火墙策略与交换机流表策略之间的新的冲突. 如果新的防火墙规则不是 Deny 规则，那么它们就不会导致被绕过的威胁. 所以，在冲突检测时我们仅考虑 Deny 规则. 在检测冲突之前，需要重新确定 Deny Authorization Space，通过检查新加入的防火墙规则和现有的防火墙规则间的依赖关系，确定新的 Deny Authorization Space. 然后我们获取与之对应的 Shifted Flowpath Space，将新的 Deny Authorization Space 和得到的 Shifted Flowpath Space 进行比较以检测冲突的发生. 此外，在更新现有防火墙规则的时候也可能会改变内部表依赖关系而导致 Deny Authorization Space 的改变. 因此，在更新防火墙规则时也应该更新 Deny Authorization Space，像添加新的防火墙规则一样，重新计算 Deny Authorization Space，然后与对应的 Shifted Flowpath Space 进行比较，检测是否存在冲突.

在添加新的防火墙规则或者更新防火墙规则时，当新的防火墙规则被添加到 Deny Authorization Space 中，相应的 Shifted Flowpath 就会建立. 在 Shifted Flowpath Space 中记录着与新的防火墙规则相关的 Flowpath 的源地址和目的地址. 而且，通过直接比较 Shifted Flowpath Space 信息和防火墙 Deny Authorization Space 信息可以检测出策略的冲突. 如果 Shifted Flowpath Space 比防火墙的 Deny Authorization Space 小，那么 Shifted Flowpath 就可以被移除. 但是如果 Shifted Flowpath Space 比新的防火墙 Deny Authorization Space 大，我们就需要阻止 Shifted Flowpath Space 中与防火墙冲突的部分信息流.

图 3 展示了一个例子，当一个新的规则被添加到防火墙中，防火墙发现 Shifted Flowpath Space 比新的 Deny Authorization Space 大. 因此防火墙通过插入更高优先级的阻断流表的方式，在头交换机和尾交换机处拒绝了 Flowpath 中冲突的那部分信息流.

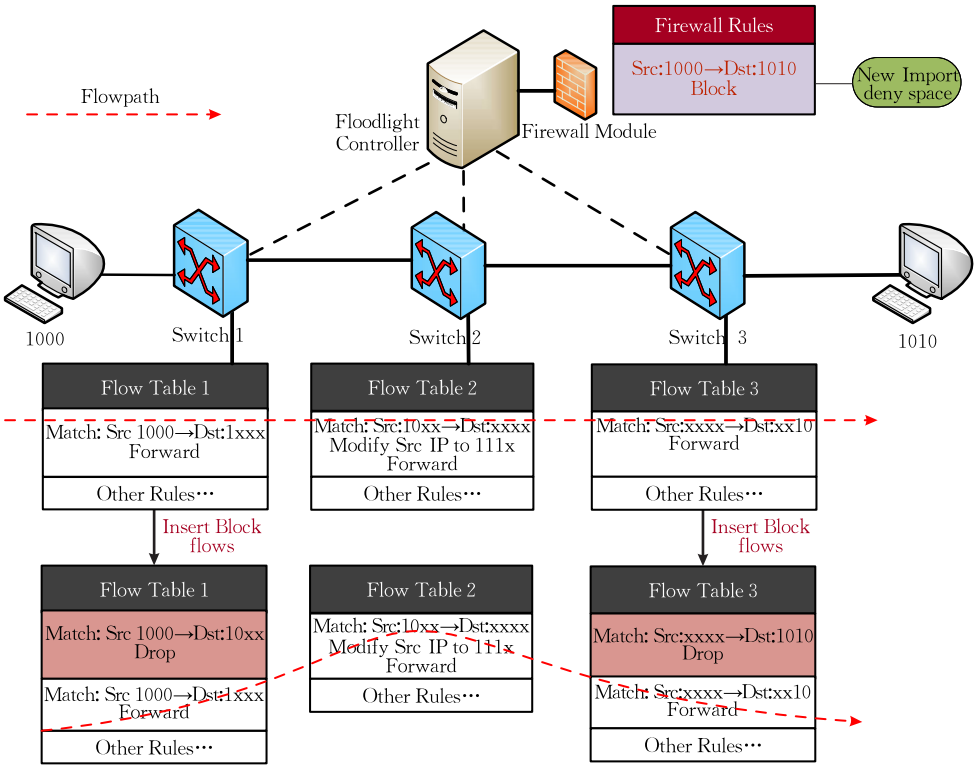

图 3 从头尾交换机添加流表解决冲突

4.3 添加流表项和更新流表项

当网络应用程序或者控制器向交换机流表中添加新的流表项或者更新流表项时，网络的状态会发生改变，从而可能引入与防火墙策略相违背的新冲突. 因此，在流表项发生改变时，需要更新流地址空间，重新计算 Shifted Flowpath. 此时没有必要再重新构造全网的 Shifted Flowpath，而只需要对插入新的流表项的交换机的那个局部 Shifted Flowpath 进行更新. 然后，我们再将新的 Shifted Flowpath Space 和 Deny Authorization Space 进行比较，检测是否存在冲突. 一旦检测出冲突，我们冲突解决模块会拒绝这些流表项的添加或者更新请求.

图 4 展示了一个例子，当一个新的流表项被插

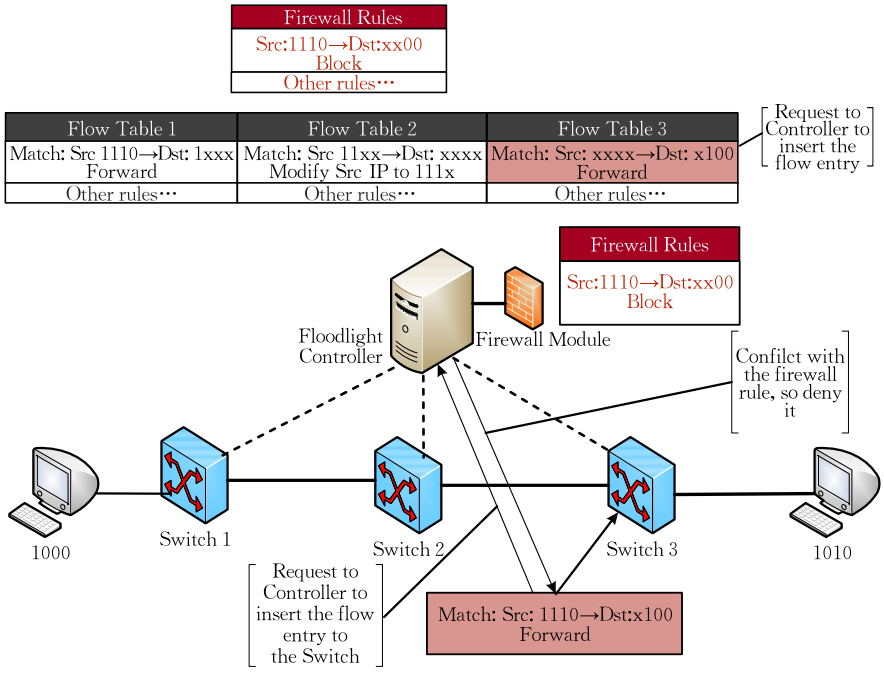

图 4 拒绝流表项添加解决冲突

入到 Table 3 中，我们检测出在防火墙策略和这条新的流表项之间存在冲突. Shifted Flowpath Space (1110 → x100) 比防火墙的 Deny Authorization Space(1110→xx00)小. 所以添加这条新的流表项的请求被拒绝. 当 Shifted Flowpath Space 比 Deny Authorization Space 大时，冲突部分将会在头尾交换机处被阻断(如图 4 所示).

5　实现与评估

5.1　实　现

我们在 Ubuntu 12.04 LTS 下使用 Mininet 模拟 SDN 网络资源，通过虚拟机中运行 Mininet 搭建网络拓扑结构. 使用 Floodlight-0.9.0 版本作为网络的后台控制器，我们在控制器中实现了 Flowpath 探测模块，冲突检测模块和冲突解决模块. Flowpath 探测模块可以构建 Shifted Flowpath. 基于 Shifted Flowpath，冲突检测模块可以通过比较防火墙的 Deny Authorization Space 和 Shifted Flowpath Space 检测安全策略的间接违反. 当检测到策略冲突时，冲突解决模块将根据不同情况进行自动化的冲突解决.

如图 5 所示，控制节点主要包含 Floodlight 及 Flowpath 探测模块、冲突检测、冲突解决模块. 这些新的功能构成了安全增强的 SDN 防火墙 FlowVerifier. 控制器通过 Java API 提供的拓扑管理服务获取、存储拓扑信息. 同时，它通过 Java API 函数读取每个 Mininet Switch 里的流表，并生成 Flowpath 图，存储起来. 我们可以通过 Floodlight 控制器平台提供的 Java API 获取所有流的信息，同时我们在控制器平台实现了一个新的 Java API 获取所有防火墙规则的信息. 当应用程序或者控制器往 Mininet Switch 里添加新的流表项时，Flowpath 图就会做相应更新. 当冲突检测模块检测到 Flowpath 和防火墙规则之间存在冲突时，它将调用冲突解决模块自动化进行冲突解决. 为了优化我们所开发的应用程序的性能，我们将源地址和目的地址转换成二进制向量. 然后，通过直接比较二进制位计算出防火墙 Deny Authorization Space 和 Shifted Flowpath Space 的交集，大大提高了冲突检测的效率.

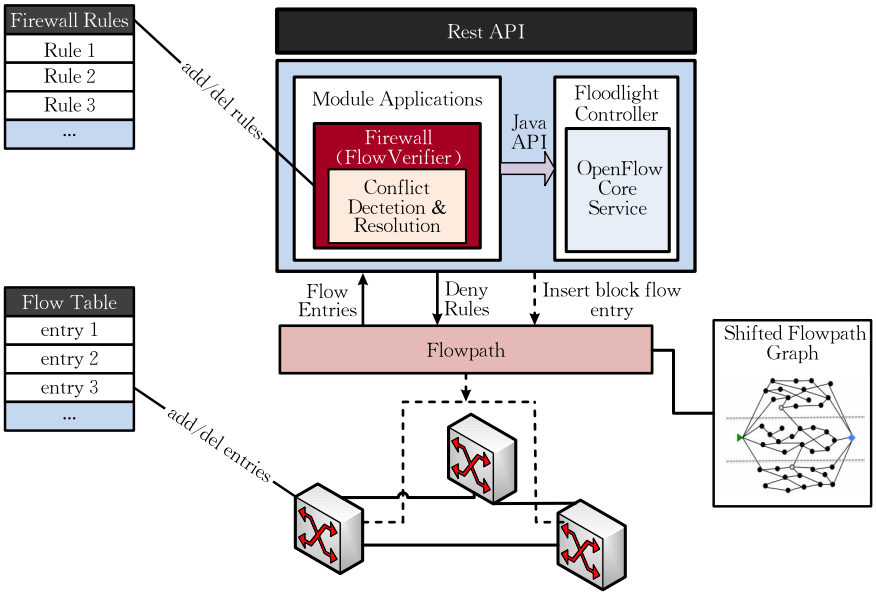

图 5　FlowVerifier 的实现框架

5.2　评　估

为了评估 FlowVerifier 的效果和性能，我们将实验系统搭在四核 Intel Core i5-3230M 2.6 GHz CPU 和 4 GB 内存的 Ubuntu 12.04LTS 主机上，并针对防火墙策略被绕过的实例 1(图 1)和实例 2(图 2)进行测试，针对这两种情况，FlowVerifier 可以找到安全策略冲突并自动解决这些冲突，检测冲突和冲突解决的耗时如表 1 所示. 冲突检测的时间包括建立 Shifted Flowpath 和比较防火墙 Deny Authorization Space 和 Shifted Flowpath Space 的时间.

表 1　FlowVerifier 在实例 1 和实例 2 中的耗时

实例	冲突检测时间/ms	冲突解决时间/ms
实例 1	2.00	3
实例 2	2.46	6

此外，我们还实现了在 FortNox[5] 中提到的别名集冲突检测方法，并将我们的方法和他们的方法

作了对比.针对上面设计两种方案,我们比较了两种方法的有效性和误报情况,结果如表 2 所示.表中所填结果的第 1 个属性值代表有效性,第 2 个属性值代表是否存在误报.根据结果显示,在实例 1 中,两种方法的检测结果都是有效的且不存在误报,但是在实例 2 中,两种方法虽然都检测出了结果,但是别名集冲突检测方法存在误报情况.当 Flow2 没有插入到交换机时,误报出现了.此时,没有从主机 A 到主机 C 的信息流,只有主机 B 到主机 C 的信息流.别名集冲突检测方法存在误报是因为它仅仅将被修改的地址加入到别名集中,将源地址集合和目的地址集合与防火墙规则进行对比.它并没有考虑到网络中的实时流状态,相反,我们的方法基于实时的 Flowpath 并且通过比较防火墙 Deny Authorization Space 和 Shifted Flowpath Space 空间,能够精确地检测策略冲突.

表 2 FlowVerifier 和 FortNox 有效性对比

方法	实例 1	实例 2
FlowVerifier	有效检测出冲突,且不存在误报	有效检测出冲突,且不存在误报
FortNox	有效检测出冲突,且不存在误报	有效检测出冲突,存在误报

考虑到网络拓扑的复杂度和流表项的数量会成为影响 FlowVerifier 系统性能的两个主要因素,我们创建了简单网络拓扑结构(图 6)和复杂网络拓扑结构(图 7).简单网络拓扑包括 3 台交换机(S1,S2,S3)和 4 台主机(h1,h2,h3,h4);复杂网络拓扑结构包括 8 台交换机和 64 台主机.这些交换机中的流表构成了 Flowpath.同时,这些交换机通过 OpenFlow 协议与控制器通信.如果交换机中的某条流表项更改了,交换机会发送一个 Flow-Removed 类型的消息到 Floodlight 控制器.因此,Floodlight 控制器可以实时地更新网络拓扑结构,与此同时,冲突检测模块也会及时检测冲突.在这个实验中,我们的目标是在不同数量的流表项情况下测试 FlowVerifier 的性能,为了实现这个目的,我们专门编写了计算并记录延迟时间的函数,将系统获取防火墙规则和所有流表项所用的时间排除在外.我们预先在防火墙模块中配置了防火墙规则,将它们分别与 100,200,300,…,1000 条候选流表项对比检测.我们在两种网络拓扑中分别进行了测试,结果如图 8 所示.从结果我们可以看出,随着候选流表项的增加,冲突检测计算所用时间线性增长.而且,我们还可以观察到,两种网络拓扑结构在耗时上的差异,通过这种差异对比,我们可以知道拓扑结构的复杂程度对于冲突检测时间是有影响的,但是我们同时也可以看到复杂拓扑与简单拓扑在添加 1000 条流表项的冲突检测时间相差仅 3 ms 左右,因此可知在较复杂网络拓扑的情况下 FlowVerifier 也能在较短时间内计算出冲突.

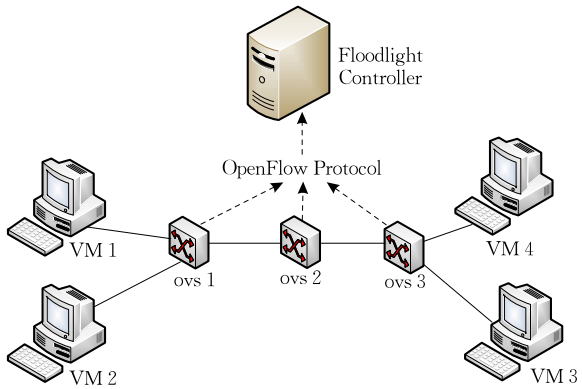

图 6 简单网络拓扑

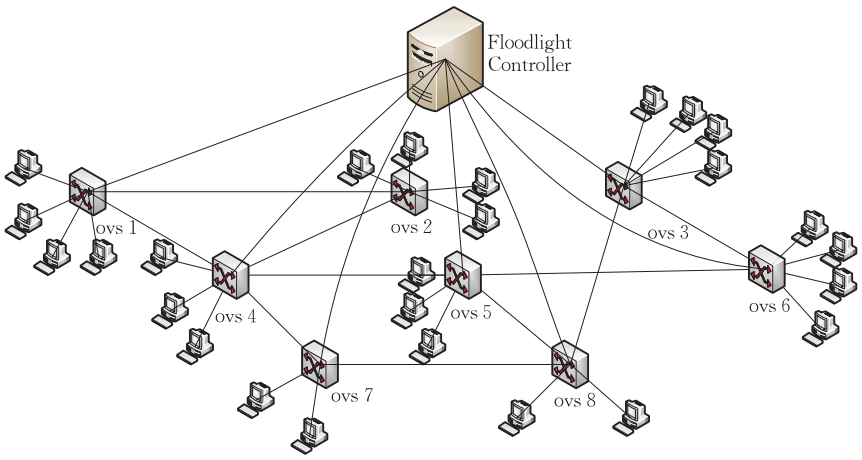

图 7 复杂网络拓扑

我们也比较了添加不同数量的候选流表项到交换机,原版 Floodlight 控制器和带有 FlowVerifier 模块的控制器所用时间,在这个实验中,我们使用网络拓扑结构为图 6 所示的简单拓扑结构,我们添加

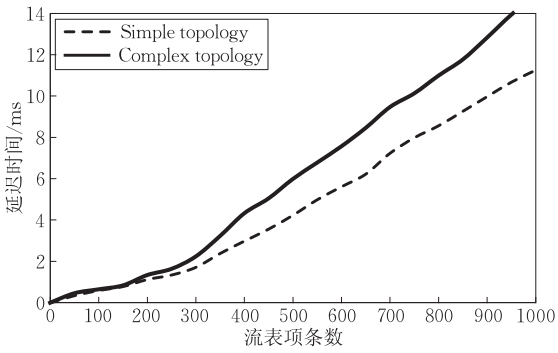

图 8　不同数量的流表项情况下 FlowVerifier 的性能

流表项的条数为 10，100，1000，分析结果如图 9 所示. 由结果我们观察到，随着流表项条数增加，添加流表时间近乎线性增加，与此同时，我们也可以观察到 FlowVerifier 模块所占用耗时也渐渐增多，但是，在流表项条数可控的情况下，延迟是在可接受范围内的.

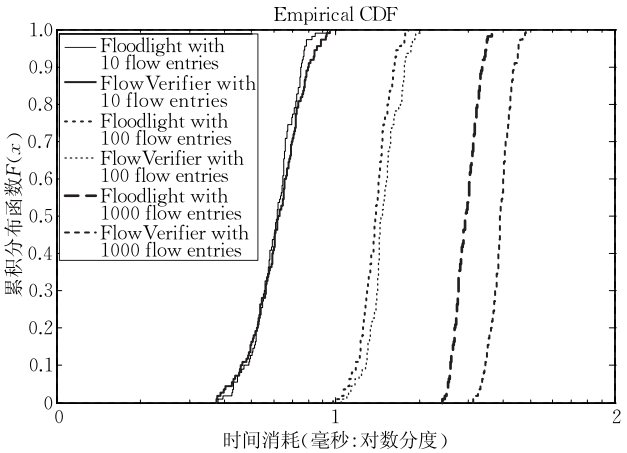

图 9　比较 Floodlight 与 FlowVerifier 添加 10，100，1000 条流表项的性能

与此同时，我们还测试了多种情况下原版 Floodlight 控制器和带有 FlowVerifier 模块的控制器在 CPU 开销上的差异，实验网络拓扑如图 7 所示，每种情形我们都做了 10 次重复实验，每次实验取最高结果，把 10 次实验数据作平均，测试结果如表 3 所示.

表 3　Floodlight 和 FlowVerifier 在多种情况下的平均最高 CPU 占用率

情景	Floodlight 平均最高 CPU 占用率/%	FlowVerifier 平均最高 CPU 占用率/%
启动时	250.3	275.6
输入 10 条流表项	12.7	14.6
输入 50 条流表项	26.4	30.5
输入 100 条流表项	50.1	51.2
输入 500 条流表项	125.9	134.6
输入 1000 条流表项	147.8	153.6

由表 3 可知，在不同状况下 Floodlight 和 FlowVerifier 的 CPU 占用率是不相同的，我们可以看到的是 FlowVerifier 相较于原版 Floodlight 在 CPU 开销上的差异不是很大.

6　相关工作

随着 SDN 技术的快速应用与发展，有关 SDN 安全话题备受关注. 在 OpenFlow 和 SDN 网络自身的安全性方面，FlowVisor[6] 建立了一个网络虚拟化平台，其网络分片技术通过在控制层和数据层增加一层，实现虚拟网络隔离从而提供一定的安全保障. FLOVER[7] 是一个可以检测部署在 OpenFlow 网络中的流表策略是否违反了网络的安全策略的新型检测系统，它可以检测出由于出错导致无效和不可见的路由，但是它并没有考虑到防火墙策略. Pyretic[8] 提出了一种高级语言用于应用程序策略的合成. 它通过一个策略合成器将防火墙策略和流表策略合成到一起，然后将合成后的策略推送到每一台 SDN 交换机的流表空间中. 该方法可以检测防火墙策略和流表策略的直接冲突，但是无法检测它们之间由于流表重写或流表间依赖关系而导致的策略间接违反. 此外，该方法将合成后的策略推送到交换机中，但是 SDN 交换机使用三态内容寻址存储器(TCAM)存储流表，其存储空间非常有限，因此当防火墙规则较多时，这种推送到每台交换机的方式是不实用的. 再者，如果流表策略完全而不是部分违反了防火墙策略，在实际当中可以直接将其丢弃，也不需要将违反的防火墙策略安装到 SDN 交换机中. Anteater[9] 利用 SAT 解析器将数据平面的信息转换成布尔表达式，然后将需要检测的策略属性转换成 SAT 问题，从而检测策略的属性，如一致性. FlowChecker[10] 将网络策略转换成逻辑表示，然后利用二进制决策图(Binary Decision Diagram)检测网络策略的属性. 但上述两种方法对网络策略的检测是静态的，没有获取和描述网络的策略的动态变化，因此无法检测防火墙策略的间接违反. VeriFlow[11] 利用图搜索技术描述网络的状态和验证网络的性质，如可达性，从而可以潜在地检查安全冲突，但是却不能自动和实时地解决冲突.

Floodlight 包含了一款开源的基于 OpenFlow 标准的 SDN 防火墙应用，但是该防火墙仅仅是把传统防火墙的包过滤功能应用到了 SDN 控制器中，仅能够检测策略的直接违反.

与我们的研究工作最相近的是 Shin 等人给出的一种安全增强的 SDN 控制器 FortNox，在 FortNox 中他们针对现有防火墙容易被绕过的问题，提出了别名集策略检测方法，该方法通过将流表中被修改过的源地址和目的地址加入到 IP 地址集合中，然后与防火墙策略中的 IP 地址进行比较来检测策略违反. 该方法能够检测出部分策略违反，但由于缺乏跟踪数据流路径的机制，从而导致该方法存在误报. 针对上述问题，我们提出了基于网络状态的实时动态策略冲突检测与解决方法. 通过获取实时的 SDN 网络状态，我们的方法能够更准确地检测防火墙策略的直接和间接违反. 同时，与 FortNox 不同，我们基于 SDN 网络状态初步给出了自动化的冲突解决方法.

7 结 论

SDN 的开放性和动态性使得网络管理变得更加灵活和智能，然而这些特性也使得 SDN 面临新的安全威胁. 本论文针对现有的 SDN 防火墙可以被通过改写交换机中的流表项轻松绕过这一安全问题，提出了基于 Flowpath 的实时动态策略冲突检测与解决方法. 通过获取实时的 SDN 网络状态，并比较防火墙 Deny Authorization Space 和 Shifted Flowpath Space，我们的方法能够准确地检测防火墙策略的间接违反，并且一旦发现冲突，可以基于 Flowpath 进行自动化和细粒度的冲突解决. 最后，我们在开源控制器 Floodlight 上实现了一个安全增强的防火墙应用 FlowVerifier，并在 Mininet 下对 FlowVerifier 进行了性能的评估. 结果表明 FlowVerifier 能够检测和自动化地解决 SDN 网络中由于流表改写而引入的策略冲突及其带来的安全威胁，并且其性能开销在可接受范围内.

参 考 文 献

[1] Wen Xitao, Chen Yan, Hu Chengchen, Shi Chao. Towards a secure controller platform for OpenFlow applications// Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN13). Hong Kong, China, 2013: 171-172

[2] Kreutz D, Ramos F, Verissimo P. Towards secure and dependable software-defined networks//Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN13). Hong Kong, China, 2013: 55-60

[3] Kazemian P, Varghese G, McKeown N. Header space analysis: Static checking for networks//Proceedings of the 9th USENIX Symposium on Network Systems Design and Implementation (NSDI). San Jose, USA, 2012: 3-5

[4] Kazemian P, Chang M, Zeng Hongyi. Real time network policy checking using header space analysis//Proceedings of the 9th USENIX Symposium on Network Systems Design and Implementation (NSDI). Lombard, USA, 2013: 4-6

[5] Porras P, Shin S, Yegneswaran V, Fong M. A security enforcement kernel for OpenFlow networks//Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN2012). New York, USA, 2012: 123-125

[6] Sherwood R, Gibb G, Yap K K, et al. FlowVisor: A network virtualization layer. OpenFlow Switch Consortium, CA, USA: OPENFLOW-TR-2009-1, 2009

[7] Son S, Shin S, Yegneswaran V, Porras P. Model checking invariant security properties in OpenFlow//Proceedings of the IEEE International Conference on Communications (ICC' 2013). Budapest, Hungary, 2013: 2-6

[8] Monsanto C, Reich J, Foster N, Rexford J, Walker D. Composing software defined networks//Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation. Berkeley, USA, 2013: 1-14

[9] Mai H, Khurshid A, Agarwal R, et al. Debugging the data plane with anteater. ACM SIGCOMM Computer Communication Review, 2011, 41(4): 290-301

[10] Al-Shaer E, Al-Haj S. FlowChecker: Configuration analysis and verification of federated OpenFlow infrastructures// Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration. Chicago, USA, 2010: 37-44

[11] Khurshid A, Zhou W, Caesar M, et al. VeriFlow: Verifying network-wide invariants in real time. ACM SIGCOMM Computer Communication Review, 2012, 42(4): 467-472

WANG Juan, born in 1976, Ph. D., associate professor. Her current research interests include cloud and SDN security, trust computing, access control and security protocol.

WANG Jiang, born in 1992, undergraduate student. His research interests include security of SDN, network architecture and cloud computing.

JIAO Hong-Yang, born in 1993, undergraduate student. Her research interest is network security.

WANG Yong, born in 1991, graduate student. His research interests include security of cloud and SDN.

CHEN Shi-Ya, born in 1993, undergraduate student. Her research interest is security of data center.

LIU Shi-Hui, born in 1990, graduate student. His research interest is security of software-defined networking

HU Hong-Xin, born in 1974, Ph. D., professor. His research interests include software-defined networking security, security and privacy in social networks, access control models and mechanisms, network and system security.

Background

Software-Defined Networking (SDN), which offers programmers network-wide visibility and direct control over the underlying switches from a logically-centralized controller, not only has a huge impact on the development of current networks, but also provides a promising way for the future development of Internet. SDN, however, also brings forth some new security challenges. First, OpenFlow protocol is almost stateless. If a host or a network device sends a flow to the network, only the first packet of the flow will be checked by the controller while the subsequent packets will be directly forwarded by the switches without any exploration. Then, an attack can make use of the Set-Field actions in flow entries to construct a malicious flow. Thus, the existing firewall security policy could be easily bypassed by inserting the flow entries with rewriting operations deliberately. Second, the dependency of flow entries can also lead to the inefficiency of security policy. The threats could affect some hosts or virtual machines in datacenters, further causing severe security issues of the whole OpenFlow-based Networks.

To address such challenges, we present FlowVerifier, a novel solution for conflict detection and resolution of SDN policies. FlowVerifier can build shifted flow paths graph in real time. Based on the shifted flow paths graph, FlowVerifier can verify the policy conflicts through checking shifted flow space against firewall authorization space. Once a policy conflict is detected, FlowVerifier can facilitate a fine-grained conflict resolution according to different conditions of conflicts. Particularly, it can automatically block a malicious flow by inserting specific blocking flow entry into ingress switch or egress switch of the flow. Finally, we present the architecture of FlowVerifier and implement an SDN application based on our proposed approach in Floodlight. We also evaluate the performance and effectiveness of FlowVerifier in a Mininet visualization network. The results demonstrate that FlowVerifier can resolve the threats of security policy violations and yet achieve acceptable performance.

This work is sponsored by the National Basic Research Program of China (973 Program) under Grant No. 2014CB340600 and the National Natural Science Foundation of China under Grant Nos. 61173138, 61402342 and 61003628.