计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(11):271−276 [doi: 10.15888/j.cnki.csa.006099]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 OpenFlow 的云计算监控架构①

常　震[1], 程绍银[2], 庄　严[2]

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230027)
[2](中国科学技术大学 信息安全测评中心, 合肥 230027)

摘　要: 云计算中网络的监控任务是一件重要和复杂的工作. 在云计算网络中的安全设备不仅要监控外网流量, 还要监控内网流量. 现有使用 SDN 和 OpenFlow 对网络流量监控的研究都基于每种设备仅存在一台的前提, 受限于安全设备的数量, 性能和效率受到限制. 本文在 OpenFlow 上提出了一种云计算网络流量监控架构和算法, 支持多台同类型的设备同时工作, 可以在云计算网络中, 实现灵活高效的网络流量监控.

关键词: 云计算; 网络; 监控; OpenFlow; 算法

引用格式: 常震,程绍银,庄严.基于 OpenFlow 的云计算监控架构.计算机系统应用,2017,26(11):271–276. http://www.c-s-a.org.cn/1003-3254/6099.html

Monitoring Framework Based on OpenFlow in Clouds

CHANG Zhen[1], CHENG Shao-Yin[2], ZHUANG Yan[2]

[1](School of Computer Science and Technology, University of Science &Technology of China, Hefei 230027, China)
[2](Information Technology Security Evaluation Center, University of Science &Technology of China, Hefei 230027, China)

Abstract: Monitoring networks in cloud is important and complex. In the cloud computing, network security equipment does not only monitor external network traffic, but also review the traffic within the network to resist attacks. There are some researches that introduce SDN and OpenFlow to route the traffic to security device. However, these methods are limited by the number of devices of the same type, and their performance is hence affected. In this paper, we propose a new flexible cloud computing monitoring framework based on OpenFlow. And we also propose the corresponding algorithm to improve the performance and efficiency. By doing so, we can monitor the network traffic efficiently and flexibly.

Key words: cloud computing; network; monitoring; OpenFlow; algorithm

1　引言

随着计算机系统的发展, 云计算[1]最近成为通过互联网托管和服务的一种新模式和趋势. 云计算集中了大量计算资源, 存储资源和软件资源, 在互联网上为客户远程提供服务, 以资源租用、应用托管、服务外包为核心, 迅速成为计算机技术发展的热点. 云计算因为其便利、经济和高扩展性的特点, 得到了教育界、产业界和政府机构的重点关注[2], 越来越多的服务开始迁移到云计算平台上. 到 2017 年, 全球公有云市场将达到 1460 亿美元, 2015 年公有云市场为 870 亿美元, 复合增长率为 22%. 云计算的高速发展带来很多值得关注的安全问题, 云计算网络中的安全监控, 成为一个重要话题.

通常, 为了对网络中的流量进行监测, 会在网络边界布置防火墙和入侵监测系统, 防止外网的入侵和木马病毒的肆虐, 对于向外网用户提供网络服务的网络设备还需要用流量信息系统防御 DDOS 进行. 租借给外部用户云计算网络中的设备使用可能被用于发起攻

① 收稿时间: 2017-03-01; 修改时间: 2017-04-05; 采用时间: 2017-04-07

© 中国科学院软件研究所 http://www.c-s-a.org.cn

击, 即攻击不仅可能来自于网络外部, 也可能来自于网络内部. 另外, 由于云计算中的网络设施数量大, 网络变动频繁, 因此网络安全设备的管理也是一个繁琐的工作. 现在有一些对云计算网络实现对网络流量监测的研究. 文献[3]提出了一种表规则描述语言, 对流量进行选择, 筛选和导向. 文献[4]提出了借助 OpenFlow 实现将流量导向对特定设备的方法. 文献[5]提出了使用策略将流量路由到安全设备. 然而这些方法都存在两点不足: (1) 对于同类安全设备数量的支持不足, 他们都假设每种类型的安全设备都有且只有一个, 在大规模的应用时, 可能存在性能瓶颈; (2) 对监控路径的计算的效率低, 计算出的结果不佳.

基于此, 本文提出了一种新的基于 OpenFlow 的高效云网络监测架构, 它基于规则和集合的方法, 用分组和集合减少规则数量, 增加对同种设备数量的支持, 同时提出相应的算法, 得到比较优秀的路径结果.

2 总体设计

本文提出的方法基于 OpenFlow. 对于特定的网络流量包, 它们将被路由到“监控网络”(monitor network). 监控网络由以下三部分组成: (1) 由网络安全设备组成的监控层, 负责对路由到它们的数据包进行监控; (2) 设备管理器, 负责注册和管理网络中的监控设备; (3) 路由计算器, 负责对符合规则的流量计算合适的路由路径.

2.1 OpenFlow 简介

OpenFlow[6]最早由斯坦福大学提出, 是最早实现的软件定义网络 (SDN)[7]的标准协议. 它将原本完全由交换机/路由器控制的数据包转发的过程, 转化为由 OpenFlow 交换机 (Switch) 和控制器 (Controller) 共同完成的过程.

在经典路由和交换机中, 数据的转发功能和路由控制规则都在相同的设备上. OpenFlow 将这两个功能分离开. 数据转发部分仍然保留在交换机 (Switch) 上, 但是交换机的路由控制功能被集中移动到独立的控制服务器 (Controller) 上, 允许在网络中使用更加复杂的流量控制和转发规则. 交换机和控制器通过 OpenFlow 协议进行通信.

在 OpenFlow 交换机的数据通道 (data path) 提供了一个抽象流量, 每个流量条目包含一组需要匹配的字段, 以及一个动作 (例如 send-out-port 或者 drop). 当 OpenFlow 交换机接收到一个流表中不存在的数据包时, 它通过 packet-in 消息将该包发送到控制器, 然后由控制器决定改分组的选择. 它可以将数据包转发到特定端口或者丢弃, 也可以为交换机添加相应的流表.

2.2 监控层

所有监控设备组成了监控层. 在之前的研究中, 往往假设对于同一类型的网络安全设备, 网络中仅存在一台. 对于这台唯一的设备, 寻找路径的算法很容易得到. 但是对于较大的网络, 一台设备服务整个网络明显是不足的, 如果网络中所有符合条件的流量都到这台监控设备, 会影响网络和这台设备的效率. 为此, 我们需要在网络的不同位置中部署同类型的设备, 而每台设备负责一片区域, 符合监控条件的流量需要遵守就近原则. 如图 1, M1 和 M2 是同类型的设备, 负责相同的工作, 但是位于不同的位置. 假设存在从 R1 到 R5 的流量需要进行监控, 最短路径是 R1→R6→R7→R5. 这里 M2 与 R6 相连, 应该选择 M2 监控该流量, 而不是 M1. 当网络中同类型的设备增多时, 如何选择合适的监控设备也成为一个问题. 我们将在后边的算法部分描述我们的方法.

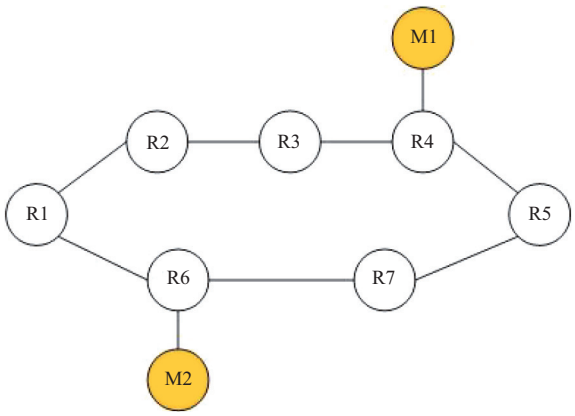

图 1 网络中存在多台被动型设备示例

为了更好的描述这个问题, 我们在这里引入设备集合 (monitor set) 的概念. 对于一个设备集合 S, S 中包含同一种功能的所有安全设备. S = {M1, M2, M3…}, 表示 M1, M2, 和 M3 都是同一种类型的设备, 但是位于网络中的不同位置. 例如, S 可以是所有 HTTP 请求计数器的集合. 对于不同的设备集合 (不同功能), 可以描述为 S1, S2 ….

我们使用一个简单的语言注册和管理网络中的安全设备. 为了注册安全设备, 需要知道设备的 ID, 设备

© 中国科学院软件研究所 http://www.c-s-a.org.cn

所属的设备集合, 设备安装的位置, 以及设备的功能.

为了更清楚的描述我们的方法, 假设网络中安装有两台相同的 HTTP 入侵检测设备 M1 和 M2, 工作在被动模式, ID 分别为 1 和 2, 它们同属于设备集合“S1”, M1 安装在路由 R4 上, M2 安装在路由 R6, 如图 1 所示, 所以 M1 和 M2 的描述分别为:

{1, S1, R4, HTTP 入侵检测}

{2, S1, R6, HTTP 入侵检测}

对于工作在主动模式下的设备, 与被动模式下不同的是安装位置的描述如图 2 中的 M3, M3 安装在 R6 与 R7 之间, 用一个元组 (R6, R7) 表示 M3 的安装位置. 所以, 对于 M3 的描述为:

{3, S1, (R6, R7), HTTP 入侵检测}

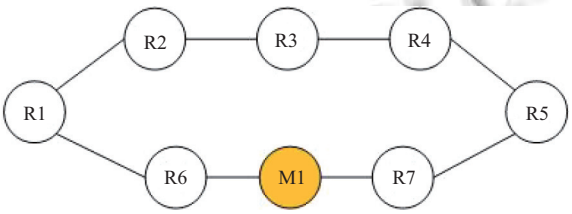

图 2 主动型监控设备示例

所有的监控设备组成了网络的监控层, 而有 OpenFlow 完成将网络流量导向到合适的监控设备. 对于同一中类型的监控设备, 需要合适的部署在网络中的不同位置, 以达到较高的效率. 对于如何部署安全设备不在本文的讨论范围以内, Sekar 等在文献[8]中提出了以中部署 NIDS 和 NIPS 的方法, 通过使用监控流量以优化资源.

3 监控规则

为了创建一个安全监控服务, 管理员需要创建一个合适的监控规则. 监控规则由两部分组成: (1) 流表条件, 表示需要监控的流量符合的规则; (2) 监控设备, 描述符合条件的网络流量应该由哪些监控设备处理. 监控规则由管理员定义以后保存在监控管理器中. 监控规则是 OpenFlow 的 match fields 中支持的规则, OpenFlow 1.0 支持包括 dl_src, dl_dst 等 15 个 match fields, 从 OpenFlow 1.2 开始, OpenFLow 支持通过 OpenFlow Extensible Match 定义的 TLV 编码的协议[9]. 我们这里使用 HTTP 入侵检测的例子, 并且使用四个 match fields(源 IP 地址和端口, 目的 IP 地址和端口).

管理员通过指定符合规则的网络流量由哪些安全设备集处理, 可以由唯一的一个设备集处理, 也可以指定多个设备集. 为了方便, 我们把这些设备集组成的集合叫做监控组. 监控组 G = {S1, S2, …}, 不同的监控组含有不同的设备集, 含有相同设备集的监控组属于同一个监控组. 监控规则可以描述为:

{*Condition* => G}

处理 HTTP 入侵监测为例, 我们已经有设备集 S1 = {M1, M2}, 我们需要将访问 192.168.0.1 的 HTTP 服务使用监控组 G1 = {S1}处理. 这里我们定义如下安全规则:

{*:*→192.168.0.1:80 => G1}

4 路由算法

为了使定义的安全规则生效, 需要使被监控的流量正确的经过安全监控设备. 使用 SDN/OpenFlow 可以使我们方便的控制网络数据的路由路径. 我们需要数据包的路由路径同时满足以下条件:

(1) 正确的到达所有要求的监控设备, 在符合安全条件的前提下, 并且能够正确到达目的地. 在我们的监控网络中, 对安全规则的监控组 G={S1…}, 每个设备集 S 至少有一个设备被符合规则的数据包经过. 传统的网络中, 只需要数据包能够正确的到达目的地. 然而传统网络路由算法, 如 OSPF[10], 不能满足我们要求经过特定安全设备的条件.

(2) 在路由路径正确的前提下, 路由路径应接近最优. 监控设备的存在, 不应该严重影响原有网络的正常工作.

(3) OpenFlow 中, 允许数据包复制往多个端口, 所以这里的路径可以产生分支, 也可以由多个路径共同组成.

(4) 对于被动型的设备 M, M 故障不影响数据包的正确到达目的; 对于主动型的设备 M, M 必须位于路径主干上, 也就是当数据包被 M 丢掉时, 无法到达目的地.

总结来说, 我们需要一条或多条路径, 使得从源端点 S 出发, 经过路由和复制, 对于安全规则中的监控组 G, 每个设备集 S 中至少有一台设备 M 接收到了该数据包, 其中被动型设备位于路径枝干上, 主动型设备位于路径主干上, 并使得该路径加权接近最小.

这里我们提出三种算法, 前两种算法针对被动型的设备, 第三种算法针对主动型的设备.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

(1) 算法 1

首先提出一种简单的使用多条路径的算法, 对路径的优化没有要求. 首先寻找从起点到终点的最短路径, 然后依次从每个设备集开始, 寻找从起点到该设备的最短路径, 选择距离最近的设备节点, 将该路径加入到总的路径中去. 最后得到多条可能存在交叉和重复的路径. 算法详细概述如下.

步骤 1. 初始化空路径 T;

步骤 2. 使用 Dijkstra 算法, 计算起点到每个设备以及终点的最短路径;

步骤 3. 将起点到终点的最短路径加入到路径 T 中;

步骤 4. For 设备集 S in 监控组 G:

选择距离起点 Src 最近的设备 M, 并将路径加入 T 中;

步骤 5. 输出路径 T.

如图 3 所示, M1 和 M2 同属于设备集 S, 按照算法 1 得到两条路径, 分别是 Src→R5→Dst, 和 Src→R6→R7→M2.

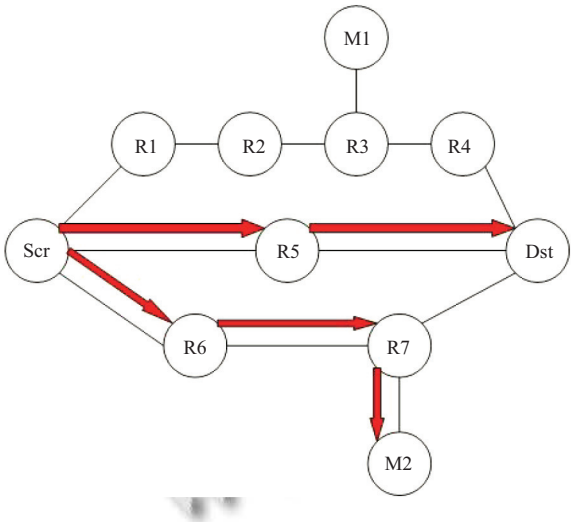

图 3 算法 1 结果示例

(2) 算法 2

算法 1 中由于不考虑路径总长度, 可能存在冗余路径, 对于网络性能影响较大. 在此提出一种修改的最小生成树算法. 具体步骤如下.

步骤 1. 初始化路径 T 和集合 $D=\{Src\}$ 和 $G=\{Dst, S1, S2\ldots\}$;

步骤 2. 使用 Dijkstra 算法, 计算起点到每个设备和终点, 每个设备之间的最短路径;

步骤 3. 选择最小的路径<U, V>, 使得 U 在集合 D 中, 设备 V 在集合 G 中, 将路径加入到 T 中, 将 U 加入到 D 中;

步骤 4. 从 G 中删除 V 所在的设备集或者 Dst, 重复步骤 3, 直到 G 集合空.

步骤 5. 输出路径 T.

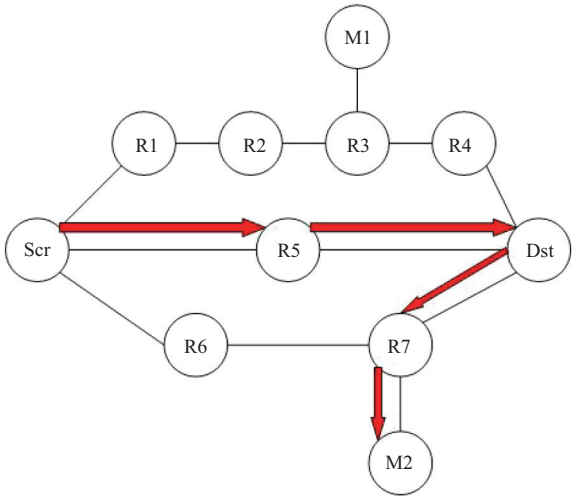

图 4 算法 2 结果示例

这里与 Prim 算法的区别在于, 每个设备 M 都属于一个设备集 S, 将 M 加入到集合 D 中后, 需要将同集合的其他设备删除, 也就是移除集合. 算法示例如图 4. 首先将路径 Src→R5→R6 加入路径 T 中, 然后得到 Dst→R7→M2 是最短路径, 加入到 T 中, 删去 M2 所属集合的所有设备{M1, M2}. 得到最终路径 T.

(3) 算法 3

算法 1 和 2 都是针对被动型的监控设备, 针对主动型的监控设备的算法要比其更麻烦. 因为要确保设备位于路径主干上, 同时还要从多个设备中选择合适的设备. 针对主动型的设备, 我们提出的算法分为两部分. 第一部分从每个集合中选择合适的设备, 第二步计算一条从 Src 开始, 经过第一部分选出来的设备, 到达 Dst 的没有回路的路径. 下边依次详细介绍详细算法.

从每个设备集中选择合适的设备的时候, 我们期待选择出来的设备生成的路径较短. 这里, 我们从集合 S 中选择设备 M, 使得从起点到 M 和从 M 到终点的最短距离之和最短. 选出来的所有设备组成集合 SC. 例如图 5, C1 和 C2 属于同一个集合, C1 到起点和终点距离之和比 C2 近, 选择 C1.

选出设备集合 SC 后, 需要计算从 Src 开始到 Dst,

© 中国科学院软件研究所 http://www.c-s-a.org.cn

经过 SC 中每个点, 且没有回路的路径. 这里我们的方法是先计算 Src 到 SC 中每点, SC 中各点之间, 以及 SC 各点到 Dst 的最短路径, 转化为旅行商 (TSP)[11]问题, 然后使用 LKH 算法求出路径, 最后消除冲突和环路. LKH[12]是对求 TSP 问题的 Lin-Kernighan 启发式算法的高效实现, 是目前 TSP 问题最好最快的解法之一. 最后, 具体步骤如下.

步骤 1. 求 Src 到 SC 中每点, SC 中各点之间, 以及 SC 各点到 Dst 的最短路径, 组成矩阵 Matrix.

步骤 2. 将步骤 1 的结果 Matrix 赋予 LKH 程序, 求出最短路径.

步骤 3. 还原步骤 2 求出的路径, 并标记环路, 找到经过两次以上的点.

步骤 4. 从起点开始对每个经过两次以上的点做处理: 将其随机分配给前一段路径, 后一段在删去此点和前一段路径经过的点后, 重新使用 Dijkstra 方法计算. 直到得到一条通路, 不存在环路.

步骤 5. 输出路径.

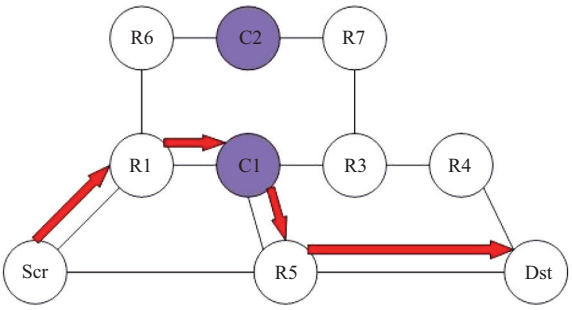

图 5 算法 3 结果示例

如图 5 示例, 最后路径是 Src→R1→C1→R5→Dst.

可以看到, 在我们的算法中, 每个算法都需要首先计算各个设备节点之间的最短路径, 当多次使用算法时, 存在大量重复计算. 因此, 我们可以首先将各节点的最短路径求出, 放入缓存中, 并设置失效时间, 当网络发生变动或者缓存中不存在时, 再重新计算最短路径. 通过缓存, 可以减少算法使用 Dijkstra 的次数, 增加算法速度.

5 实验与测试

为了检验本文的方法, 我们在 Mininet 中对文中提到的三种算法进行了测试. Mininet 是轻量级的软件定义网络平台, 支持使用 OpenFlow 协议对网络进行管理. 我们使用了一个有 10 个路由节点的网络, 两个被动型设备组 S1 和 S2, 分别有两台虚拟设备挂靠在不同的路由上. 同时, 还使用了一组两台主动型的虚拟设备. 平台使用 1 G 内存, Intel Core i5-6300HQ 2.30 GHz 单核的 Ubuntu 14.10 虚拟机. 并使用不经过安全节点的最短路径对比. 在未使用缓存的情况下, 结果如表 1.

表 1 路径生成结果对比

方法	路径生成时间(ms)	路径长度
最短路径	0.2	2
算法1	0.6	6
算法2	2.1	5
算法3	0.7	4

经过测试, 三个算法中 Dijkstra 求最短路径的过程占用的时间达到 98%. 这是因为要计算每对设备之间的最短路径的计算量较大. 当使用缓存后, 三个算法的时间消耗接近相同. 但是算法 2 的路径长度要优于算法 1.

6 结语

在这篇文章中, 我们介绍了一种基于 OpenFlow 的云计算监控架构, 不受限于网络中安全设备的数量. 并提出了三个算法, 两个对应被动型监控设备, 一个对应主动型监控设备, 可以在不影响网络正常工作的前提下, 对云计算网络的流量进行高效灵活的监控.

云计算网络的安全是一件重要的工作, 在接下来的研究中, 我们将结合云计算网络存在的具体威胁, 使用本文提出的方法, 对相应的攻击提出对应的策略和机制. 例如 SDN 网络中存在的网络投毒污染[13]和对 Controller 服务器的 DOS 攻击[14]也值得我们关注, 需要采取合适的方法进行抵御.

参考文献

1 Mell P, Grance T. The NIST definition of cloud computing. National Institute of Standards and Technology, US Department of Commerce, 2011.

2 冯登国, 张敏, 张妍, 等. 云计算安全研究. 软件学报, 2011, 22(1): 71–83. [doi: 10.13328/j.cnki.jos.005010]

3 Ballard JR, Rae I, Akella A. Extensible and scalable network monitoring using OpenSAFE. Proc. of the 2010 Internet Network Management Conference on Research on Enterprise Networking. San Jose, CA, USA. 2010, 8.

4 Shin S, Gu GF. CloudWatcher: Network security monitoring

© 中国科学院软件研究所 http://www.c-s-a.org.cn

using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). Proc. of the 2012 20th IEEE International Conference on Network Protocols (ICNP). Austin, TX, USA. 2012. 1–6.

5 Raza S, Huang GY, Chuah CN, *et al*. Measurouting: A framework for routing assisted traffic monitoring. IEEE/ACM Trans. on Networking, 2012, 20(1): 45–56. [doi: 10.1109/TNET.2011.2159991]

6 Mckeown N, Anderson T, Balakrishnan H, *et al*. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69–74. [doi: 10.1145/1355734]

7 Haleplidis E, Pentikousis K, Denazis S, *et al*. Software-defined networking (SDN): Layers and architecture terminology. Internet Research Task Force, 2015.

8 Sekar V, Krishnaswamy R, Gupta A, *et al*. Network-wide deployment of intrusion detection and prevention systems. Proc. of the 6th International Conference. Philadelphia, Pennsylvania, USA. 2010. 18.

9 OpenFlow switch specification version 1.2. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf, 2011.

10 Moy J. RFC 2178 OSPF version 2. 1997.

11 Hoffman KL, Padberg M, Rinaldi G. Traveling salesman problem. Encyclopedia of Operations Research and Management Science. 3rd ed. New York, NY, USA. 2013. 1573–1578.

12 Helsgaun K. An effective implementation of the Lin-Kernighan traveling salesman heuristic. European Journal of Operational Research, 2000, 126(1): 106–130. [doi: 10.1016/S0377-2217(99)00284-2]

13 Hong S, Xu L, Wang HP, *et al*. Poisoning network visibility in software-defined networks: New attacks and countermeasures. Network and Distributed System Security Symposium 2015. San Diego, CA, USA. 2015.

14 Shin S, Gu GF. Attacking software-defined networks: A first feasibility study. Proc. of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. Hong Kong, China. 2013. 165–166.

www.c-s-a.org.cn

© 中国科学院软件研究所 http://www.c-s-a.org.cn