

# 基于贝叶斯-斯坦科尔伯格博弈的 SDN 安全控制平面模型

卢振平, 陈福才, 程国振

(国家数字交换系统工程技术研究中心, 河南 郑州 450002)

**摘要:** 提出一种基于动态异构冗余的安全控制平面, 通过动态地变换异构的控制器以增加攻击者的难度。首先, 提出基于贝叶斯-斯坦科尔伯格博弈模型的控制器动态调度方法, 将攻击者和防御者作为博弈参与双方, 求得均衡解, 进而指导调度策略; 其次, 引入一种自清洗机制, 与博弈策略结合形成闭环的防御机制, 进一步地提高了控制层的安全增益; 最后, 实验定量地描述了基于该博弈策略的安全控制层相比与传统部署单个控制器以及采用随机策略调度控制器的收益增益, 并且自清洗机制能够使控制平面一直处于较高的安全水平。

**关键词:** 软件定义网络; 网络安全; 贝叶斯-斯坦科尔伯格博弈; 控制器

**中图分类号:** TP309

**文献标识码:** A

**doi:** 10.11959/j.issn.2096-109x.2017.00211

## Secure control plane for SDN using Bayesian Stackelberg games

LU Zhen-ping, CHEN Fu-cai, CHENG Guo-zhen

(National Digital Switching System Engineering & Technological R&D Center, Zhengzhou 450002, China)

**Abstract:** A dynamic scheduling controller in SDN control layer was proposed by dynamically transform heterogeneous controlled in order to increase the difficulty of the attacker. Firstly, a dynamic scheduling method based on Bayesian Stackelberg games the attacker and defender were game participation on both sides, obtained the equilibrium, which guided the scheduling strategy. Secondly, introducing a self-cleaning mechanism, it improved the gain of the control layer security combined with game strategy form closed-loop defense mechanism. The experiments described quantitatively based on the game strategy compared with traditional safety control layer to deploy a single controller and adopt the strategy of random scheduling profit gain of the controller, and self-cleaning mechanism could make the control plane to be in a higher level of security.

**Key words:** software defined networking, network security, Bayesian Stackelberg games, controller

## 1 引言

软件定义网络 (SDN, software defined network) 将控制与转发分离, 形成如图 1 所示的应

用层、控制层、数据层, 通过南向可编程接口开放了网络, 促进了网络创新, 简化了网络管理。控制器作为其核心组件, 掌握着整个网络的视图, 将上层决策翻译为数据面的转发规则, 一旦被攻

收稿日期: 2017-05-10; 修回日期: 2017-09-20。通信作者: 卢振平, 13203728376@163.com

基金项目: 国家自然科学基金创新群体基金资助项目 (No.61521003); 国家重点研发计划基金资助项目 (No.2016YFB0800100, No.2016YFB0800101); 国家自然科学基金青年基金资助项目 (No.61309020); 国家自然科学基金青年基金资助项目 (No.61602509)

**Foundation Items:** The Foundation for Innovative Research Groups of the National Natural Science Foundation of China (No.61521003), The National Key R&D Program of China (No.2016YFB0800100, No.2016YFB0800101), The National Natural Science Foundation of China (No.61309020), The National Natural Science Foundation of China (No.61602509)

击或劫持,可导致信息泄露,甚至使整个 SDN 网络瘫痪<sup>[1]</sup>。

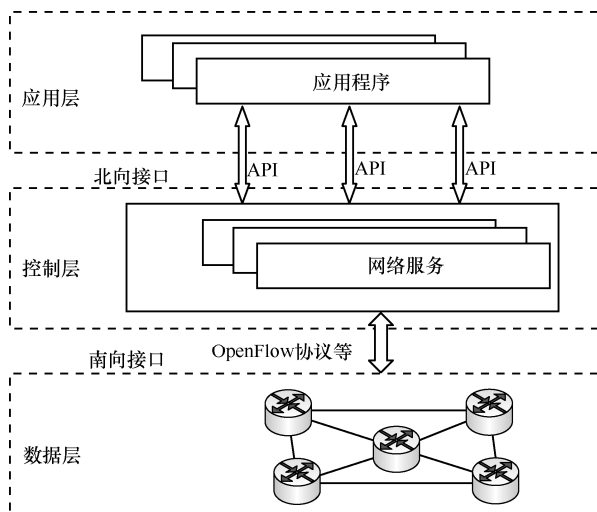


图 1 SDN 架构

一般地,针对控制器的攻击来自 2 个方向。首先,攻击可能来自上层应用或管理员,恶意应用或管理员利用控制器漏洞发动攻击。例如,一个简单的 exit 命令可致使 Floodlight 退出造成网络瘫痪<sup>[2]</sup>。其次,攻击可能来自底层数据面。攻击者通过南向接口发送探测报文,探测控制器的状态和身份信息,然后利用其掌握的控制器的漏洞,进一步地发动劫持、信息篡改、甚至 DoS 等攻击<sup>[3]</sup>。例如,文献[4]通过开放的 OpenFlow 协议,控制 SDN 网络中的 1 台主机,发送探测报文,获取控制器的状态信息,为后续利用控制器的漏洞发动攻击提供了有效信息。总之,利用控制器的漏洞发动攻击是控制器面临的主要威胁之一<sup>[5]</sup>。

为了应对控制器面临的潜在威胁,研究人员提出了大量方法。从架构上,由单控制器<sup>[2,6,7]</sup>转变为分布式控制<sup>[8-11]</sup>、弹性分布式控制<sup>[12,13]</sup>。该方法可有效抵御控制器的单点失效问题,尤其是弹性分布式控制器可以通过调整控制器实例数量,应对动态变化的网络负载。但是主流的分布式架构中不同控制器实例仍是同构的,因此,包含的漏洞也是重叠的,当某个重叠漏洞被利用时,攻击事件可影响整个分布式控制平面。利用拜占庭容错机制的控制层<sup>[14,15]</sup>避免了单点失效,然而,由于控制器之间是相互交互的,有可能造成攻击感染。从防御机制上,约东南向和北向 2 个接口,

通过北向应用接口的分级调用<sup>[16]</sup>、南向接口的限制访问<sup>[17]</sup>,有效避免了接口的滥用,但是该方法具有侵入性,需要重新设计接口,甚至开发新的控制器内核,仍无法避免控制器内部的漏洞利用。目前,已有研究未从防御控制器安全漏洞的角度出发,无法避免漏洞利用攻击。

本文基于动态异构冗余的思想,设计出一种动态调度多样控制器的控制层,包括多个异构控制器集合以及调度器等核心组件,通过调度器从集合中动态地选择控制器,以改变传统控制平面相对静态的特性,从而降低控制器脆弱点被利用的影响。鉴于攻防特点,为了最大化策略的有效性,将控制器的选择问题建模为贝叶斯-斯坦科尔伯格博弈模型,攻击者和防御者作为博弈双方参与者,刻画收益函数,求得均衡解,从而指导调度控制器;其次,为了进一步提高防御者的安全性,引入自清洗机制,从而形成一个闭环的防御机制。

## 2 相关工作

SDN “三层两接口”结构在方便管理的同时也引入了新的脆弱性,控制器安全问题作为 SDN 典型安全威胁重要的一环,逐渐成为制约 SDN 发展的关键因素<sup>[18]</sup>。目前,学术界针对控制层的脆弱点研究主要分为两方面。

### 1) 传统加固式安全

Phillip Porras 将开源控制器 Floodlight 扩展为 SE-Floodlight<sup>[8]</sup>,增加应用程序证书管理模块、安全审计子防御者、权限管理等新的安全模块,对控制器安全进行审计和跟踪。FortNOX<sup>[19]</sup>架构是针对开源控制器 NOX 设计的一种安全内核,在 NOX 控制器上分别增加了基于角色的数据源认证模块、状态表管理模块、流规则冲突分析模块和流规则超时回调模块等安全功能,提高流规则转发的安全性。Shin 等<sup>[7]</sup>通过对常见控制器的安全性进行测试和分析后,设计了一种新型、内嵌安全机制的集中式控制器 Rosemary。其内核将所有应用程序封装在一个应用程序区内,实时监控应用程序的行为,并通过检查各个应用程序的签名信息判定其是否合法。OFX<sup>[20]</sup>是一种新的网络安全应用模型,在现存 OpenFlow 架构下,通过

将安全应用机制打包下放至底层网络设备,实现更高效的安全应用的开发。这些单纯的加固式安全无法应对不断爆出的安全漏洞导致的单点失效,也无法保证控制层与数据层之间数据完整性、机密性等。

## 2) 分布式控制器

分布式控制平面解决了单控制器情况下单点失效的威胁,主要包括静态分布式控制平面和动态分布式控制平面。其中,静态分布式控制平面交换机和控制器的映射是静态配置的。HyperFlow<sup>[8]</sup>是 OpenFlow 的首个分布式控制平面,作为应用程序运行于安装 NOX 的控制器之上,负责控制器网络状态视图的同步(通过传播本地产生的控制事件),将到达其他控制器管理的交换机的 OpenFlow 命令重定向到目标控制器上,并重定向来自交换机的响应到发起请求的控制器。FlowVisor<sup>[9]</sup>在转发平面与控制平面之间引入了一个软件切片层,并采用 OpenFlow 实现了该切片层,用以切分控制消息。Kandoo<sup>[10]</sup>以及 Onix<sup>[11]</sup>也属于静态分布式控制平面。

由于静态配置导致负载均衡分布不均,Advait Dixit 等设计了 ElastiCon<sup>[12]</sup>,一种弹性分布式控制器体系,在该体系中,控制器池中控制器的数量可以根据流量状态和负载动态地增加或减少。ElastiCon 为了完成负载迁移,提出了一种新的交换迁移协议,在分布式控制器与交换机之间采用动态映射以更大程度地增强网络的扩展性。在 ONOS<sup>[13]</sup>和 ODL<sup>[21]</sup>分布式体系结构中,当一个组件或实例失败,且有其他剩余实例,允许重新分配,保障防御者仍能继续工作。

然而,这些分布式控制器实例的同构性可能存在共有的安全漏洞,一旦被攻击者探测利用便可以发动持续地攻击,劫持控制平面,进而破坏整个网络。因此,文献[14,15]提出基于拜占庭容错机制的控制器安全解决方案。文献[14]中每个底层交换机由多个控制器控制,控制器数量根据不同的容错需求变化,能够抵抗对控制器的拜占庭攻击以及控制器和交换机通信之间的攻击,避免了单点失效,保证了网络的正常运行。文献[15]设计出一种拜占庭容错的分布式控制器,使网络能够容忍控制平面以及数据平面的错误,并整合

了现有 2 种易受拜占庭错误攻击的开源控制器,实现了容错机制。然而这种方法一旦一个控制器被攻击成功,有可能造成攻击感染。文献[22]提出了一种控制器和交换机的动态分配算法以最小化控制平面的响应时间,然而,该动态分配控制器是为了更优的负载均衡,并未考虑安全。

本文借鉴已有的研究成果,针对 SDN 中控制器漏洞的典型安全威胁,为了解决控制器本身脆弱性带来的安全问题,提出一种动态调度控制器的 SDN 安全控制层<sup>[23]</sup>,综合考虑通过动态调度相互隔离的异构控制器降低攻击者探测成功的概率,并且避免了利用控制器漏洞和后门,进而持续高效地进行恶意攻击。

## 3 防御模型及原理分析

### 3.1 SDN 安全控制层模型

本文设计了一种 SDN 安全控制层,在控制平面引入动态异构冗余因子,在多个不同的防御者维度上,开发、分析和部署防御者可控的、随时间动态地移动和变化的机制和策略,以限制自身脆弱性的暴露,增加攻击者的攻击成本,降低被攻击机会。

图 2 给出了动态调度 SDN 控制器的基本结构。在应用层和数据层,与现有的 SDN 架构的数据平面功能一致。在动态控制层,包含功能等价结构相异的  $N$  个控制器执行体集合。

控制器异构度:各类控制器采用的语言各不相同,Ryu 采用 Python 语言实现,ONOS 以及 Floodlight 采用 Java 语言实现,但 ONOS 基于 OSGI 框架实现,这与 Floodlight 相异。语言相异以及实现方式的不同是衡量异构的指标。

控制器实例在角色上包括两类:一类是 Master(主)控制器,有且仅有一个,是实际上管理网络的控制器;一类是 Slave(备)控制器,作为备选的资源池。数据代理将底层的网络请求通过此分发至控制器集合,控制器的管理控制数据也经此下发给底层网络等;感知器感知数据平面的网络安全状态并将信息交互到调度器。调度器包含所有控制器的注册信息,接受感知器的信息,内有倒计时功能,一旦计时器清零,调度器启动调度策略选择主控制器。

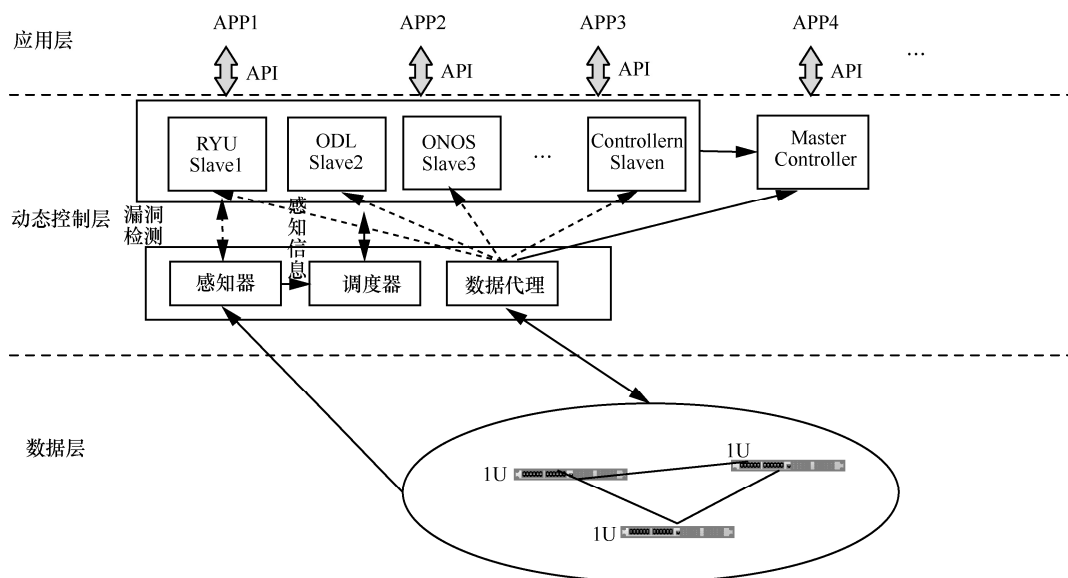


图 2 安全控制层的 SDN 架构

如图 2 所示，控制平面包括  $N$  个相互独立异构的控制器，依据调度策略从备选资源池选一个主控制器控制底层网络，本文设计了 2 种不同的调度周期设置机制，流程如图 3 所示。

1) 主动式调度机制：调度器在时间轴上的调度间隔一致，调度器定时自发地对控制器进行切换。

2) 反应式调度机制：感知器模块感知控制层的安全状态，若发现异常情况，则立即依据调度策略进行调度。

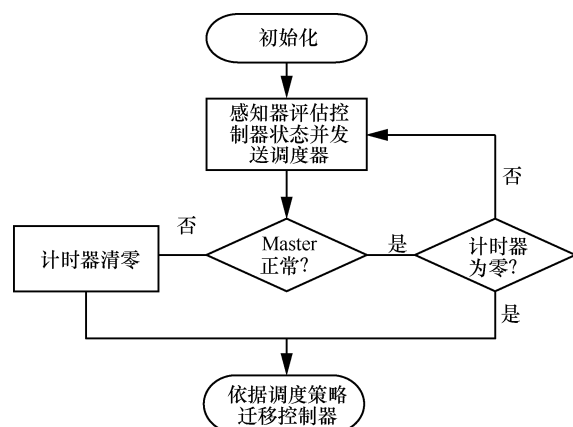


图 3 动态调度控制器流程

图 4 为安全架构的闭环防御机制。调度器指导控制层切换控制器之后，控制层会依据历史攻击信息，评估控制层的安全状态<sup>[24]</sup>，利用本身的安全资源开始自清洗，修复脆弱点，再进行新一轮的调度。以一个简单的用例说明动态调度的安全效果。

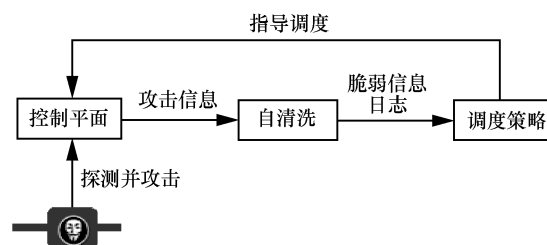


图 4 闭环防御机制

攻击行动：使用 Java 语言基于 OSGI 框架的 ONOS 控制器也带来了 OSGI 的脆弱性。如图 5 所示，当交换机收到一个没有匹配项的数据分组时，交换机就会把这个数据分组发送给控制器。当处理不规则、不完整、恶意制定的数据分组时，ONOS 的解串器会引发异常，如果不及时发现并处理这个异常，那么相关交换机就会与 ONOS 断开连接。远程攻击者可以利用这个漏洞实施 DoS 攻击造成 ONOS 与交换机断开连接。攻击者只需要发送恶意数据分组就可以利用这个漏洞，这是一个通过数据层面利用控制器漏洞的典型案列。

防御机制：在动态调度的控制器架构中，启用调度机制，Ryu 或 Floodlight 成为主控制器时，远程攻击者利用此 ONOS 的漏洞失效，即攻击行动无效。攻击者必须重新探测阶段获取新的脆弱点组织攻击。

随机选择备选资源池里的控制器是最简单的策略形式。然而，随机策略相对盲目，没有考虑执行体获取的攻击的历史信息，如不同能力的攻

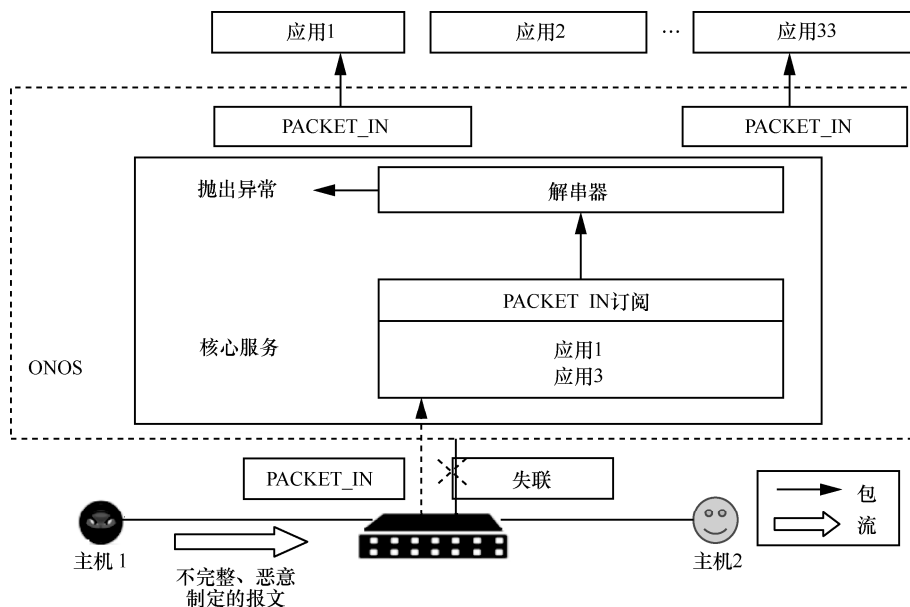


图5 ONOS 控制器攻击实例

击组织以及攻击的倾向性。实际上，攻防双方均对对方有一定的信息支撑<sup>[25]</sup>。所以，在攻防双方均有一定先验知识的情况下，防御者选择主控制器的调度策略与整个控制层的安全性休戚相关。接下来，在3.2节研究控制器调度策略问题，在3.3节详述自清洗机制。

### 3.2 博弈论分析

贝叶斯-斯坦科尔伯格博弈<sup>[26]</sup>是一种典型的解决探测以及攻击不确定性的安全博弈方法，本节将攻防过程建模为贝叶斯-斯坦科尔伯格博弈模型，获得最佳的控制器调度策略。

#### 3.2.1 博弈预备知识

1) 斯坦科尔伯格博弈：通常用来刻画安全领域的攻防探测场景，包括先采取行动的领导者以及随后行动的追随者。由于领导者首先采取行动，所以领导者在此博弈中占优<sup>[26]</sup>。以一个简单的博弈例子说明。

如图6所示，纵列是领导者策略 $S_{den} = \{a, b\}$ ，横列是追随者策略 $S_{att} = \{c, d\}$ ，该博弈的纯策略纳什均衡是 $([a], [c])$ ，此时领导者收益为2。对于领导者来说，策略 $b$ 是严格占主导地位的，如果领导者在追随者之前选 $b$ ，则获得一个为3的收益，因为追随者为了确保自己最高收益会选 $d$ 策略。如果领导者采取随机策略，即分别以0.5的概率选 $a$ 和 $b$ ，则追随者将会选策略 $d$ 应对，以

最大化自己的收益，此时领导者将会获得一个3.5的收益。一般来说，混合策略一般要优于纯策略。

| 领导者 \ 追随者 | $c$  | $d$  |
|-----------|------|------|
| $a$       | 2, 1 | 4, 0 |
| $b$       | 1, 0 | 3, 2 |

图6 斯坦科尔伯格博弈领导者、追随者收益值

2) 贝叶斯博弈：参与者包括 $N$ 个代理，每个代理都可能有多种不同的类型。

安全领域中，通常假设领导者单一，而追随者可能包含多种类型，于是拓展斯坦科尔伯格博弈的领导者 and 跟随者为贝叶斯博弈的2个代理，领导者只有一种类型，追随者包含多个类型即为贝叶斯-斯坦科尔伯格博弈。2个代理均有各自的策略集 $S_i$ 以及收益函数 $U_i$ 。该博弈为一种完美信息博弈，目标是假定追随者知道领导者的策略，领导者找到一个最优的混合策略最大化自己的收益，此策略即被称为强斯坦科尔伯格均衡<sup>[26]</sup>。

#### 3.2.2 模型映射

动态调度控制器的防御者看作领导者（单一类型），攻击者为追随者（多种类型，如不同的攻击组织）。控制层的调度过程即选择策略的过程：控制层以一个时间周期在各个控制器之间进行切换，从一种控制器到另外一种控制器可以看作是防御者的策略。另一方面，攻击方包括很多攻击类型，

选择攻击行动是博弈中追随者的一种策略。表 1 是本文模型的符号说明。

| 表 1      | 变量说明                     |
|----------|--------------------------|
| 符号       | 意义                       |
| $t$      | 攻击类型                     |
| $l$      | 防御策略                     |
| $F^t$    | $t$ 的攻击策略集               |
| $a^t$    | 攻击类型 $t$ 的收益值            |
| $R^t$    | 防御者在 $t$ 的收益矩阵           |
| $C^t$    | 攻击者在 $t$ 的收益矩阵           |
| $p^t$    | 攻击类型 $t$ 的攻击可能性          |
| $n_f^t$  | 攻击类型为 $t$ 、策略为 $f$ 的布尔值  |
| $C_{lf}$ | 防御者策略 $l$ 、攻击者策略 $f$ 的收益 |
| $y$      | 攻击策略分布                   |
| $T$      | 攻击类型集合                   |
| $L$      | 防御者纯策略集合                 |
| $F$      | 攻击纯策略集合                  |
| $x$      | 防御者策略分布                  |

**假设** 攻击者和防御者的目标都是最大化自己的收益

**定义** 贝叶斯-斯坦科尔伯格博弈模型为四元组  $G_{BSG} = (L, T, F, U)$

1)  $L = (l_1, l_2, \dots, l_n)$  是博弈中防御者的策略集合，显然， $n$  越大，冗余度越高，动态性效果越显著。

2)  $T = (t_1, t_2, \dots, t_n)$  是攻击类型的集合， $T \neq \emptyset$ 。

3)  $F = (F_1, F_2, \dots, F_n)$ ，对任意的  $t \in n$ ， $F^t \neq \emptyset$ ， $F^t = (f_1, f_2, \dots, f_h)$ ，表示每个攻击类型至少有一种以上的策略，即  $h \geq 1$ 。不同的攻击类型的  $F^t$  允许有交集。

攻击者的攻击行动包括 2 种：1) 针对目前控制器已知的未解决的脆弱点的攻击；2) 依据已有的信息预测控制器的脆弱点的攻击。一种攻击行动对应一个漏洞，每种攻击类型包含若干种攻击行动。攻击者确定攻击行动依赖于以下 2 点。

① 攻击难度。例如，一段任意的代码造成远程代码执行攻击比利用控制器的数据库漏洞更难；代码量更少的 Ryu 比 ONOS 曝出的安全漏洞更少<sup>[27]</sup>。

② 控制器流行度。在商业网络中部署越广泛

的控制器，则攻击者对其的攻击倾向性更高。例如，相比于小范围部署的控制器，攻击者青睐于诸如对谷歌数据中心部署的 Onix 控制器的漏洞挖掘等攻击。

4)  $U = (R, C)$  是收益函数集合。 $R$  代表防御者的收益， $C$  代表攻击者的收益，对于给定的攻击者类型  $t$ 、攻击概率和防御者策略，双方期望收益如下。

$$R(x, p^t, t) = \sum_{l \in L} \sum_{f \in F} R_{lf}^t x_l p_f^t \quad (1)$$

$$C(x, p^t, t) = \sum_{l \in L} \sum_{f \in F} C_{lf}^t x_l p_f^t \quad (2)$$

防御者与攻击者收益依赖于以下 2 点。

① 对目标控制器造成的影响，包括攻击者探测数据的探针数量以及这些数据的关键性。例如，利用 ONOS 曝出的 Root 权限的漏洞攻击获取权限之后比中间人窃取数据更为严重。

② 攻击是否被检测。攻击之前探测的数量越多，被防御者检测到的概率越大。

在本博弈中，参与者双方需要满足约束条件。

1) 防御者策略约束

$$\begin{aligned} \sum_{l \in L} x_l &= 1 \\ x_l &\in [0 \dots 1] \end{aligned} \quad (3)$$

防御者以概率分布选择控制器，概率和为 1。

2) 攻击策略约束

$$\begin{aligned} \sum_{f \in F^t} n_f^t &= 1 \\ n_f^t &\in \{0, 1\} \end{aligned} \quad (4)$$

对于防御者确定的混合策略部署控制层，每个攻击类型面对的是同一个线性收益问题，攻击者存在一个最优策略，故每种攻击类型都会选择同一种纯策略回应<sup>[26]</sup>。

3) 攻击者收益约束

$$\begin{aligned} 0 &\leq (a^t - \sum_{l \in L} C_{lf} x_l) \leq (1 - n_f^t) M \\ a^t &\in \mathbb{R} \end{aligned} \quad (5)$$

$M$  是一个很大的数。当  $n_f^t = 0$ ，任何时候右边的不等号都是不成立的，当  $n_f^t = 1$  时，左边的不等式使攻击收益最大化，当攻击策略最大化攻击的收益时，两边取等号。

由以上可知，通过博弈将控制层的调度策略问题转化为一个混合整数二次规划问题，求解目

标函数下的  $x$  即为调度策略, 如下。

$$\begin{aligned}
 & \max_{x,n,a} \sum_{l \in L} \sum_{t \in T} \sum_{f \in F^t} p^t R_{lf}^t x_l n_f^t \\
 & s.t.: \sum_{l \in L} x_l = 1 \\
 & \sum_{f \in F^t} n_f^t = 1 \\
 & 0 \leq (a^t - \sum_{l \in L} C_{lf} x_l) \leq (1 - n_f^t) M \\
 & x_l \in [0 \cdots 1] \\
 & n_f^t \in \{0, 1\} \\
 & a^t \in \mathbb{R}
 \end{aligned} \quad (6)$$

在求解这个优化问题时,  $R_{lf}$  以及  $C_{lf}$  确定如下。

- ①  $V_{R_c}$  为控制器  $c$  自我估价。
- ②  $V_{Att_c}$  为攻击者对攻击控制器  $c$  的估价。
- ③  $r_c$  代表防御者认为发现攻击者的收益。
- ④  $r_{att}$  代表攻击者认为被抓到的代价。
- ⑤  $p_c$  为在部署控制器  $c$  的时候防御者抓捕

攻击者的概率。

对应策略组合  $(l, f)$  如下。

- ① 防御者收益:  $R_{lf} = -V_{R_c}(1 - p_c) + p_c r_c$ 。
- ② 攻击者收益:  $C_{lf} = V_{Att_c}(1 - p_c) - p_c r_{att}$ 。

从而获得贝叶斯-斯坦科尔伯格博弈的策略性博弈的正规表示:  $[R', C']$ 。一轮博弈之后, 防御者可依据环境以及攻击信息重新调整收益矩阵, 并且防御者依据攻击图分析能够预测不同攻击者的攻击概率以及攻击能力, 防御者已知这些信息之后, 通过求解式(6), 求得均衡策略。式(6)是一个 NP 难问题, 本文利用 DOBSS 算法求解<sup>[26]</sup>。

### 3.3 自清洗机制

防御者在一次博弈之后利用 3.2 节获得的混合策略部署控制层, 攻防过程并没有结束, 防御者根据历史博弈信息, 进入自清洗阶段。一次博弈之后的防御者根据搜集的攻击行动信息, 分析控制器脆弱点, 自清洗即防御者利用安全资源修复这些脆弱点<sup>[24]</sup>, 更新收益矩阵, 继续下一次博弈, 每一次博弈是独立的, 存在唯一的强斯坦科尔伯格均衡, 这种闭环防御机制, 使防御者一直维持在一个较高的安全状态。本节研究的问题是, 在防御者的安全资源有限的条件下, 自清洗过程首先应该解决哪些力所能及的漏洞。

假设攻击者的安全资源仅能修复一个清洗出的漏洞, 该漏洞修复后, 对应的攻击行动则需要从收益矩阵中去除, 之后在新的收益矩阵下进行下一次博弈, 移除某种攻击行动之后最收益增益大所对应的漏洞, 即关键的漏洞, 该脆弱点应该首先被修复。据此, 本文提出一种自清洗算法, 该自清洗算法通过从收益矩阵中去除若干攻击行动之后获得新的收益矩阵, 计算防御者增益, 进而比较衡量各种漏洞的关键性。若将某个攻击行动移除时计算收益最高, 则认为对应的安全漏洞应该首先被修复。而且漏洞被修复的收益矩阵作为下一次博弈的输入。

#### 算法 1 自清洗算法

**输入** 防御者策略:  $X$   
 攻击类型:  $L$   
 概率分布:  $P$   
 攻击策略:  $F$   
 收益矩阵:  $[R', C']$

**输出** 关键漏洞  $\psi$

- 1) 防御者根据安全资源确定能够解决的漏洞个数  $k$ 。
- 2) 获得  $\Psi = C_n^k$  种排列组合,  $n$  为漏洞总数。
- 3) 计算移除每种排列组合后的防御者增益。
- 4) 记录增益最大的漏洞组合  $\psi$ 。
- 5) 返回  $\psi$ 。

通过算法 1 确定控制器中的关键漏洞, 防御者根据自身有限的安全资源确定所要解决安全漏洞的数量 ( $C_n^1$ 、 $C_n^2$  等), 以使整个控制层的安全状态持续保持较高的水平。

## 4 仿真评估

### 4.1 仿真环境

本次实验在配置 Intel(R) Core(TM) i7-4790 CPU 3.6 GHz, 16 GB RAM 的主机上进行, 优化问题的求解以及自清洗算法均通过 Python 编程实现, 利用 Gurobi 优化器求解式(6)的博弈优化问题, 获得控制器最优调度策略, 并定量地计算出相比随机策略和纯策略的收益增益。

本文以在控制层中部署 4 种多样的控制器为实例, 说明如下。

防御者可以选择 Ryu、ODL、ONOS、Floodlight 这 4 种控制器作为 4 种策略, 攻击者可

能会有 9 种攻击行动。记  $A_1$ 、 $A_2$ 、 $A_3$ 、 $A_4$  是针对 4 种控制器安全漏洞的攻击行动，其中， $A_1$ 、 $A_2$  是以探测和获取数据为目的， $A_3$ 、 $A_4$  则是以劫持控制器破坏网络防御者为目的。 $A_5 \sim A_8$  表示具有 2 种攻击的效果， $A_9$  表示攻击行动无效。收益表中前者代表防御者收益，后者代表攻击收益。显然，2 种以劫持破坏控制器的攻击行动攻击者比获取数据的收益更高。如果两方的收益都为正，代表虽然攻击者企图攻击成功，但是对防御者关键数据和服务并没有造成实质的破坏，而且防御者可以从此次攻击获得一些有效信息。例如，攻击行动没有匹配防御者的漏洞，所以攻击不会成功，可攻击者仍然可以获得防御者某些配置信息，进而开始建立一个新的成功的攻击，但同时，防御者也会从未成功的攻击中获益。通过蒙特卡罗抽样以及分段常值函数近似方法<sup>[25,28]</sup>对攻防收益矩阵评估，如表 2 所示，攻击类型以及对应的攻击能力先验概率如表 3 所示。

表 2 防御者策略和攻击策略对应的收益关系

| 策略                | 攻防收益    |         |         |            |
|-------------------|---------|---------|---------|------------|
|                   | Ryu     | ODL     | ONOS    | Floodlight |
| $A_1$             | -2, 6   | 5, -8   | 5, -8   | 2, -6      |
| $A_2$             | -5, 5   | 2, -6   | -2, 6   | 5, -8      |
| $A_3$             | 5, -2   | -10, 10 | -10, 10 | 5, -2      |
| $A_4$             | -10, 10 | 5, -2   | 5, -2   | -10, 10    |
| $A_5: A_1 \& A_3$ | -8, 8   | -4, 8   | 2, 6    | 8, -8      |
| $A_6: A_2 \& A_3$ | -4, 8   | -8, 8   | 8, -8   | 2, 6       |
| $A_7: A_1 \& A_4$ | 2, 6    | 8, -8   | -8, 8   | -4, 8      |
| $A_8: A_2 \& A_4$ | 8, -8   | 2, 6    | -4, 8   | -8, 8      |
| $A_9$ : 无         | 0, 0    | 0, 0    | 0, 0    | 0, 0       |

表 3 攻击者类型的攻击能力以及攻击概率

| 攻击类型 | 攻击能力                      | 攻击概率 |
|------|---------------------------|------|
| 黑客 1 | $A_1, A_2, A_9$           | 0.5  |
| 黑客 2 | $A_1, A_3, A_5, A_9$      | 0.2  |
| 黑客 3 | 所有                        | 0.1  |
| 黑客 4 | $A_1, A_2, A_3, A_4, A_9$ | 0.2  |

表 3 表示每种攻击类型的攻击概率以及攻击行动集合。这里包括 4 种攻击类型：黑客 1~黑客 4，每一个攻击类型可能有一种或者多种攻击行动，并且每种类型的攻击行动存在交集。

## 4.2 策略分析评估

本实验利用分解式最优贝叶斯-斯坦科尔伯格求解器求解贝叶斯-斯坦科尔伯格博弈问题，使用 8 个线程，迭代了 1 589 次，用时 3 ms，优化误差在  $1 \times 10^{-4}$  以内。通过求解优化问题，得到最优的混合策略 (0.3, 0.16, 0.27, 0.27)，即防御者以此概率分布选择控制器 Ryu、ODL、ONOS、Floodlight 作为主控制器部署控制层，所获的收益最大，最大收益是 -0.62。当使用随机策略时，即以 (0.25, 0.25, 0.25, 0.25) 部署所获得的收益是 -1.25。以纯策略部署控制器 Ryu、ODL、ONOS、Floodlight，收益分别为 -6、-3、-5.6、-6。从而表明贝叶斯-斯坦科尔伯格博弈获得的调度策略优于随机策略和纯策略。

进一步地，通过改变攻击概率分布，观察防御者在 2 种情况的收益变化，比较均衡策略与随机策略所获收益的优劣，进而证明博弈均衡策略占优的一般性。为了方便比较，只考虑 3 种攻击类型：黑客 1、黑客 2 和黑客 3，结果如表 4 所示。

表 4 博弈策略和随机策略的收益对比

| 概率分布        | 随机策略的收益 | 博弈均衡策略的收益 | 收益增益 |
|-------------|---------|-----------|------|
| 0,0,1       | -2.5    | -0.85     | 1.65 |
| 0,0,2,0.8   | -2      | -0.54     | 1.46 |
| 0,0,4,0.6   | -1.5    | -0.15     | 1.35 |
| 0,0,6,0.4   | -1      | 0.22      | 1.22 |
| 0,0,8,0.2   | -0.5    | 0.61      | 1.11 |
| 0,1,0       | 0       | 1         | 1    |
| 0.2,0,0.8   | -2.5    | -0.7      | 1.8  |
| 0.2,0.2,0.6 | -2      | -0.48     | 1.52 |
| 0.2,0.4,0.4 | -1.5    | -0.27     | 1.23 |
| 0.2,0.6,0.2 | -1      | 0.05      | 1.05 |
| 0.2,0.8,0   | -0.5    | 0.53      | 1.03 |
| 0.4,0,0.6   | -2.5    | -0.32     | 2.18 |
| 0.4,0.2,0.4 | -2      | -0.1      | 2.1  |
| 0.4,0.4,0.2 | -1.5    | 0.11      | 1.61 |
| 0.4,0.6,0   | -1      | 1         | 2    |
| 0.6,0,0.4   | -2.5    | 0.06      | 2.56 |
| 0.6,0.2,0.2 | -2      | 0.57      | 2.57 |
| 0.6,0.4,0   | -1.5    | 1.5       | 3    |
| 0.8,0,0.2   | -2.5    | 1.07      | 3.07 |
| 0.8,0.2,0   | -2      | 2         | 4    |
| 1,0,0       | -2.5    | 2.5       | 5    |

一般来说, 基于该博弈模型的混合策略不会劣于随机策略和纯策略。本节则进一步定量地刻画该博弈相比随机策略和纯策略的增益值。在表5中, 防御者采用随机策略平均收益是-1.45, 而使用贝叶斯-斯坦科尔伯格博弈均衡策略的平均收益是0.37。因此, 无论攻击类型的概率如何分布, 贝叶斯-斯坦科尔伯格博弈均衡策略一般优于随机策略。

### 4.3 自清洗机制分析

$A_1 \sim A_4$  是分别对应安全漏洞  $v_1 \sim v_4$ , 通过3.2节方法找到关键的漏洞。需要说明的是, 若  $v_1$  被解决, 则攻击  $A_1$ 、 $A_4$ 、 $A_5$  都将被移除。

表5和表6分别是选取解决1到2个关键漏洞之后的收益关键结果。由表6可看出, 如果防御者只有修复一个安全漏洞的资源能力,  $v_4$  为最关键的漏洞,  $v_4$  修复之后, 收益增益最大, 为1.163。如表7所示, 如果资源能力可以修复2个安全漏洞, 则解决的关键漏洞为  $v_3$ 、 $v_4$ , 修复  $v_3$ 、 $v_4$  之后防御者的收益最高, 收益增益为1.449, 因此采取自清洗机制之后, 根据防御者的安全资源修复关键漏洞之后采取该博弈策略之后, 收益远高于最初的-0.62。

表5 解决一个关键漏洞的收益更新 (关键部分)

| 解决一个漏洞 | 收益更新   |
|--------|--------|
| $v_1$  | -0.768 |
| $v_2$  | -0.493 |
| $v_3$  | 0.318  |
| $v_4$  | 0.543  |

表6 解决2个关键漏洞的收益更新 (关键部分)

| 解决2个漏洞     | 收益更新   |
|------------|--------|
| $v_1, v_2$ | -0.757 |
| $v_1, v_3$ | 0.318  |
| $v_1, v_4$ | 0.428  |
| $v_2, v_3$ | 0.318  |
| $v_2, v_4$ | 0.557  |
| $v_3, v_4$ | 0.829  |

## 5 结束语

分布式异构控制器架构针对控制器本身以及开放的南北接口有可能被攻击者利用的脆弱点, 能够有效地防止利用这些脆弱点实施APT攻击,

尤其在一些关键的已商业部署的SDN网络等。同时, 在提高安全性的同时, 异构控制器的动态调度不可避免地引入了额外的开销代价, 如动态调度控制器之后, 控制器需要重新学习底层网络的端口、主机信息等, 以重新部署网络功能, 从而导致用户在此期间产生新的流请求服务无法上交到控制器处理, 造成损失。

控制器作为SDN架构的核心, 安全地位与日俱增。本文创新地设计了一种SDN安全控制层架构防御安全漏洞威胁, 通过基于贝叶斯-斯坦科尔伯格博弈求得防御者最优调度方案, 进而指导动态调度控制器。同时, 为了更进一步提高控制层的安全性, 本文提出了一种自清洗算法。通过实验定量地说明了博弈策略方案相比于纯策略和随机策略安全增益, 自清洗机制使控制层一直维持在较高的安全状态。本文证明了动态调度控制器的应用概念, 未来工作将以研究计算调度周期的问题模型, 均衡成本和安全需求的最优调度周期为重点。

### 参考文献:

- [1] KREUTZ D, RAMOS F M V, VERISSIMO P. Towards secure and dependable software-defined networks[C]//ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. 2013: 55-60.
- [2] SHIN S, SONG Y, LEE T, et al. Rosemary: a robust, secure, and high-performance network operating system[C]//ACM Conference on Computer and Communications Security. 2014: 78-89.
- [3] LENG J, ZHOU Y, ZHANG J, et al. An inference attack model for flow table capacity and usage: exploiting the vulnerability of flow table overflow in software-defined network[J]. Water Air & Soil Pollution, 2015, 85(3): 1413-1418.
- [4] SONCHACK J, AVIV A J, KELLER E. Timing SDN control planes to infer network configurations[C]//ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization. 2016.
- [5] LEE S, YOON C, SHIN S. The smaller, the shrewder: a simple malicious application can kill an entire SDN environment[C]//ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization. 2016.
- [6] MORZHOV S, ALEKSEEV I, NIKITINSKIY M. Firewall application for Floodlight SDN controller[C]//The International Siberian Conference on Control and Communications. 2016.
- [7] SHIN S, SNOS Y, LEE T, et al. Rosemary: a robust, secure, and high-performance network operating system[J]. 2014: 78-89.
- [8] TOOTOONCHIAN A, GANJALI Y. HyperFlow: a distributed control plane for OpenFlow[C]//Internet Network Management Conference on Research on Enterprise Networking. 2010: 3.

- [9] SHERWOOD R, GIBB G, YAP K K, et al. FlowVisor: a network virtualization layer[J]. 2009.
- [10] YEGANEH S H, GANJALI Y. Kandoo: a framework for efficient and scalable offloading of control applications[C]//The Workshop on Hot Topics in Software Defined Networks. 2012:19-24.
- [11] KOPONEN T, CASADO M, GUDE N, et al. Onix: a distributed control platform for large-scale production networks[C]//Unix Symposium on Operating Systems Design and Implementation(OSDI 2010). 2010:351-364.
- [12] DIXIT A, FANG H, MUKHERJEE S, et al. Towards an elastic distributed SDN controller[C]//The 1st Workshop on Hot Topics in Software Defined Networking (HotSDN 2013). 2013: 7-12.
- [13] BERDE P, GEROLA M, and HART J, et al. ONOS: towards an open, distributed SDN OS[C]//The Workshop on Hot Topics in Software Defined Networking. 2014:1-6.
- [14] LI H, LI P, GUO S, et al. Byzantine-resilient secure software-defined networks with multiple controllers in cloud[C]//2014 IEEE International Conference on Communications (ICC 2014). 2014: 695-700.
- [15] ELDEFRAWY K, KACZMAREK T. Byzantine fault tolerant software-defined networking (SDN) controllers[C]//IEEE Computer Society International Conference on Computers, Software & Applications. 2016:208-213.
- [16] LEE C, SHIN S. SHIELD: an automated framework for static analysis of SDN applications[C]//ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization. 2016:29-34.
- [17] WILCZEWSKI. Security considerations for equipment controllers and SDN[C]//2016 IEEE International Telecommunications Energy Conference (INTELEC). 2016:1-5.
- [18] AHMAD I, NAMAL S, YLIANTTILA M, et al. Security in software defined networks: a survey[J]. IEEE Communications Surveys & Tutorials, 2015, 17(4): 1.
- [19] 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
- [20] SONCHACK J, AVIV A J, KELLER E, et al. Enabling practical software-defined networking security applications with OFX[C]//Network and Distributed System Security Symposium. 2016.
- [21] MEDVED J, VARGA R, TKACIK A, et al. OpenDaylight: towards a model-driven SDN controller architecture[C]//IEEE International Symposium on World of Wireless, Mobile and Multimedia Networks. 2014:1-6.
- [22] WANG T, LIU F, GUO J, et al. Dynamic SDN controller assignment in data center networks: Stable matching with transfers[C]//IEEE Conference on Computer Communications. 2016:1-9.
- [23] LU Z P, CHEN F C, et al. Poster: a secure control plane with dynamic multi-NOS for SDN[C]//NDSS Posters. 2017.
- [24] LEE S, YOON C. DELTA: a security assessment framework for software-defined networks[C]//Network and Distributed System Security Symposium. 2017.
- [25] KIEKINTVELD C, MARECKI J, TAMBE M. Approximation methods for infinite bayesian stackelberg games: Modeling distributional payoff uncertainty[C]//The 10th International Conference on Autonomous Agents and Multiagent Systems. 2011:1005-1012.
- [26] PARUCHURI P, PEARCE J P. Playing games for security: an efficient exact algorithm for solving bayesian stackelberg games[C]//The 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'08). 2008: 895-902.
- [27] DAVID Jorm. 44CON LONDON 2015 Presentations[EB/OL]. <https://44con.com/44con-london-2015/44con-london-2015-presentations/>
- [28] PITA J, JAIN M, TAMBE M, et al. Robust solutions to stackelberg games: addressing bounded rationality and limited observations in human cognition[J]. Artificial Intelligence, 2010, 174(15): 1142-1171.

#### 作者简介:



卢振平(1992-), 男, 河南商丘人, 国家数字程控交换系统工程技术研究中心硕士生, 主要研究方向为软件定义网络、网络先进防御。



陈福才(1974-), 男, 江西南昌人, 硕士, 国家数字程控交换系统工程技术研究中心研究员, 主要研究方向为电信网关防、网络安全。



程国振(1986-), 男, 山东菏泽人, 博士, 国家数字程控交换系统工程技术研究中心助理研究员, 主要研究方向为云数据中心、软件定义网络、网络安全。