文章编号:1671-4229(2021)04-0063-13

SDN 架构下 ARP 攻击的解决方案及示例

徐大伟[1,2]，戴　铖[2]，祝烈煌[1*]，厍怡婕[2]
(1. 北京理工大学 网络空间安全学院，北京　100081；2. 长春大学 网络安全学院，吉林 长春　130022)

摘　要：网络上的安全一直是一个突出的问题,在本地网络中,ARP 协议工作在较底层,安全性经常被忽略,并且容易受到黑客攻击。在新的网络架构 SDN 架构中,ARP 攻击仍然是一个严重的网络安全问题,而且在这种网络架构中,不仅网络终端节点会暴露在攻击的风险下,控制器也容易遭受到攻击。因此,文章提出基于 SDN 架构的一种 ARP 攻击的解决方案。该方案通过在控制器中开发一个防御模块,在终端机中开发一个信息验证软件,利用控制器对网络信息的掌控,对要进行 ARP 通信的终端机进行验证,验证通过就代替目标终端机回复 ARP 应答报文,验证不通过就向管理员报告 ARP 攻击。实验表明,该方案能够有效防御各种形式的 ARP 攻击,有效地提高了局域网的安全性。

关键词：ARP 攻击；SDN 架构；网络安全；控制器
中图分类号：TP 393.08　　**文献标志码**：A

Solutions and examples of ARP attacks under SDN architecture

XU Da-wei[1,2], *DAI Cheng*[2], *ZHU Lie-huang*[1*], *SHE Yi-jie*[2]
(1. School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China;
2. College of Cybersecurity, Changchun University, Changchun 130022, China)

Abstract: Security of a network has always been a prominent issue. In a local network, the ARP protocol works at a lower level, and the security is often overlooked, and it is vulnerable to hacker attacks. In the new network SDN architecture, ARP attacks are still a serious network security problem, and in this network architecture, not only the network terminal nodes will be exposed to the risk of attacks, but also the controllers. Therefore, this article proposes an ARP attack solution based on the SDN architecture. This solution develops a defense module in the controller and an information verification software in the host, and uses the controller to control network information to verify the host that needs ARP communication. If the verification is passed, the target host will reply to the ARP response message instead of the target host. If the verification fails, the ARP attack will be reported to the administrator. Experiments show that the scheme can effectively defend against various forms of ARP attacks and effectively improve the security of a local area network.

Key words: ARP attack; SDN architecture; network security; controller

人们的生活与网络的联系越来越紧密,然而由于传统网络架构的控制层与转发层紧耦合,各个设备呈分布式管理,网络行业的创新十分困难[1]。因此,SDN 架构应运而生,这是一个控制层与数据层松耦合、管理式集中的架构,并且由于 SDN 可编程的特点,更有利于推进网络行业的创新。

但是 SDN 下的 ARP 攻击仍然存在,并且在这种新型网络架构[2]中,所面临的风险形势更加严峻[3]。因为 SDN 架构下的 ARP 攻击不仅会攻击终端机,而且会攻击控制器的节点。但是在已有的网络中提出解决方

作者简介：徐大伟(1977—),男,副教授,博士研究生. E-mail：xudw@ccu.edu.cn
* 通信作者. E-mail：liehuangz@bit.edu.cn

引文格式：徐大伟，戴铖，祝烈煌，等. SDN 架构下 ARP 攻击的解决方案及示例[J]. 广州大学学报(自然科学版),2021,20(4):63-75.

案无法防止SDN中控制层遭受攻击的风险[4-6]。而且现有的SDN架构下防止ARP攻击的方案极少并且都有缺陷,文献[7]中方案不能自动判断ARP攻击[7],文献[8]中方案无法防止攻击者2次接入网络的攻击[8],因此,提出一种新架构下的解决方案极为重要。

本文阐释了SDN网络的架构构成,并将ARP攻击划分为2种方式,而本文所提方案解决了SDN架构下这2种ARP攻击方式的攻击问题,基于控制器和终端机编写了两段程序,在虚假的ARP数据包到达目标终端机之前被控制器检测出并丢弃,同时控制器不会被ARP攻击所欺骗。本文最大的创新在于使用了RSA非对称密钥算法对终端机进行验证,并且使用Montgomery算法[9]对RSA进行优化,使得即使素数P、Q非常大也可以正常运行。

1 SDN架构与ARP欺骗

1.1 SDN架构

软件定义网络(Software Defined Networking,缩写为"SDN")是美国斯坦福大学Nick McKeown教授团队提出的一种新型网络创新架构[10],它将网络的控制平面和数据平面解耦分离[11],是实现高带宽、动态网络的理想架构,有利于网络行业的创新。

SDN的基本思想是路由控制平面和数据转发平面互相分离,数据平面更为通用,不再关注网络协议的转发,只需要接收控制平面的命令并进行转发[12];路由转发平面被从各个设备中剥离到一个统一的外部控制器,控制器掌握着所有网络的信息[13-14],需要接收数据层面发送的消息,并指引数据平面进行消息的转路径发送。SDN对整个网络的资源的调度和管控性是传统网络不能相比的。

SDN架构按层次结构划分,如图1所示,包含5个层次,即应用程序层、北向接口层、控制层、南向接口层和数据层[15-16]。

SDN应用程序层主要由网络管理员或其他网络研究人员用来部署一些网络应用程序,例如,负载均衡和访问控制等[14]。北向接口层为其上层提供网络的笼统视图,使用户有机会根据自己的需求开发适当的应用程序,根据网络条件规划资源。控制层由控制器组成[17],主要有3个任务:①将SDN最上层的需求转交到软件定义网络的交换机;②为软件定义网络应用提供底层网络的视图;③对转发层面进行转发的调节、自定义管理。南向接口层有如下功能:控制传输、查询设备性能、报告统计和通知事件等,可以理解为数据平面的一个编程接口。数据层主要功能是执行来自控制层的流规则,并应答控制器的指令[18-19]。

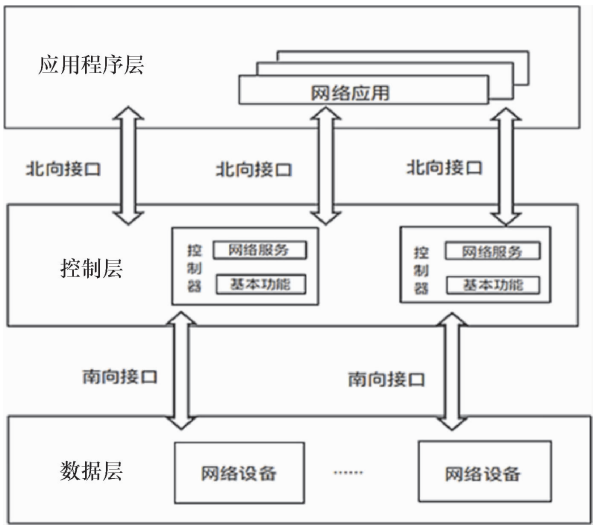

图1 SDN架构层次[1]

Fig. 1 SDN architecture levels

1.2 OpenFlow协议

OpenFlow协议是SDN架构南向接口协议标准,该协议定义了交换机传输计划的某些功能组件[20],定义了交换机的工作机制和交换机的管控机制,同时也定义了控制器和交换机在通信过程中的消息格式和类型[21]。

流在协议中的定义是一个报文的集合,即在某一个时间段内,经过某一网络,具有相同属性,并按照一定次序发送的报文的集合[22]。其中流表是OpenFlow协议交换机中的转发表[23]。协议交换机中的流表项等效于在传统网络中集成所有级别的网络配置信息,因此,在转发数据时使用更多样的规则。

OpenFlow交换机的数据包处理流程为:数据包进入交换机,交换机中的协议解析模块解析数据包包头域,然后将解析后的结果与对应的流表进行匹配[24]。到了流表内部,解析后的结果与每个流表项进行比较,如果匹配成功,则根据表项上的动作进行处理,否则丢弃或转发给控制器以请求指令[25-26]。

1.3 SDN架构下的ARP攻击形式

(1)攻击者在网络层或链路层的ARP攻击

攻击者在TCP/IP的网络层面,窃取合法用户的终端机的IP地址或者MAC地址[24],使其他终端机和控制层面错误地将攻击者的IP地址误判为对应合法终端机的MAC地址,或者错误地将攻击者的MAC地址误认为对应的合法终端机的IP地址[27-28]。

(2)攻击者同时利用网络层和链路层的 ARP 攻击

攻击者在网络层和链路层同时窃取合法用户终端机的 IP 地址和 MAC 地址,以欺骗控制层面产生错误的 MAC 地址、IP 地址与接入位置映射信息[29]。

这 2 种攻击形式在传统的 ARP 攻击解决方案中并没有进行划分,只有在文献[3]中对这 2 种形式有具体的区分。

2　方案设计

2.1　方案设计思路

方案设计思路如图 2 所示。终端机在接入网络,获得 IP 地址之后需要向控制器注册验证信息,其中包括终端机的 MAC 地址和 RSA 公钥。同时控制器会将 IP 地址、MAC 地址和接入端口号等做成验证信息表项,放在以交换机身份 ID 为索引的信息表中。在通信中,终端机发出 ARP Request 包[30],到达交换机后找不到对应

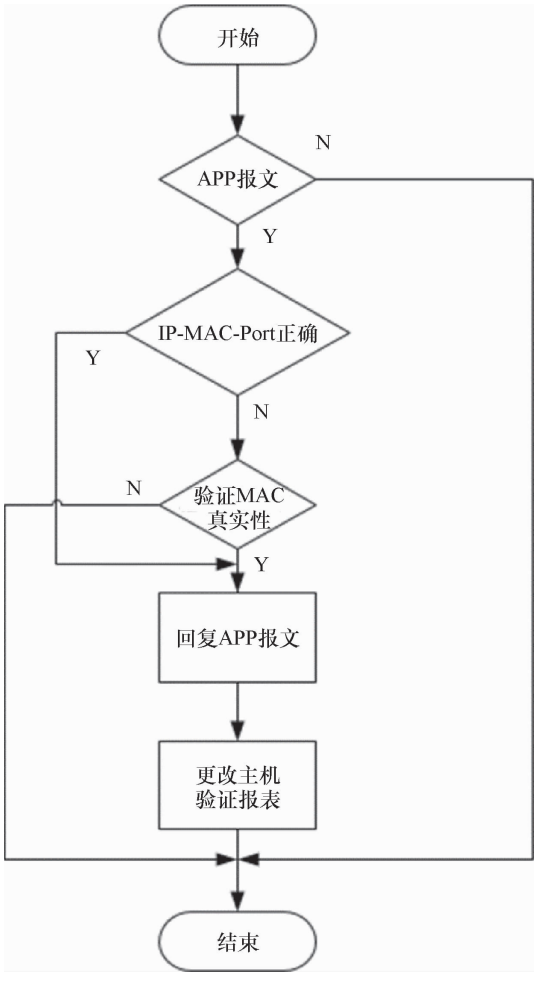

图 2　方案设计基本思路

Fig. 2　Basic idea of scheme design

流,就会把该数据包上发控制器。控制器从 Packet_in 消息中获得 ARP 数据包[31]之后,将该包的 IP 地址、MAC 地址映射与终端机的信息表项进行比对,如果映射对应不正确,则向 DHCP 服务器请求最新映射,如果还匹配不成功,则向管理员上报 ARP 攻击;如果对应成功则再将收到该数据包时的端口与匹配成功的表项对比,如果端口也匹配成功则代替目标终端机回答该 ARP 报文,如果端口匹配不成功,则记录此次 ARP 请求的目标地址并以该报文 MAC 地址对应的公钥加密一条随机消息发回该终端机,如果该终端机的 MAC 地址真实,将会回答出正确的信息,交换机会将请求的 IP 地址所对应的 MAC 地址通过消息返回给终端机,并且更新该终端机的信息。

总之,本方案中,需要每个终端机向控制器注册信息,而控制器代替目标机回复 ARP 请求,并且只有通过验证的 ARP 请求才能收到回复。

2.2　详细方案

本文所提方案要求控制器端拥有检测 ARP 消息正确与否的能力,同时也需要终端机拥有检测到控制器发送的验证信息并恢复的能力。方案主要包括 2 个主要阶段:验证信息交换阶段和通信中攻击防御阶段。

(1)验证信息交换阶段主要分为 5 个步骤

步骤 1:终端机生成 RSA 公钥和私钥。每个终端机有属于自己的 2 个大素数 P,Q,这 2 个大素数为自己的特征值,终端机利用自己的特征值计算出公钥和私钥,将公钥存在本地,RSA 算法流程图如图 3 所示。

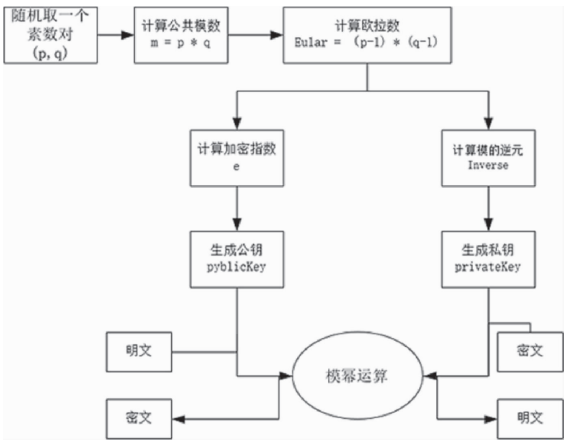

图 3　RSA 算法流程图

Fig. 3　RSA algorithm flow chart

步骤 2:终端机与控制器交换公私钥及其它信息。终端机通过一个 ARP 消息向控制器发送注册消息,包括自身的 IP 地址、MAC 地址和最初接入端口以及自己的公钥,其中终端机公钥封装在 ARP 消息的 payload 域

中。带有 payload 域的 ARP 消息如图 4 所示。

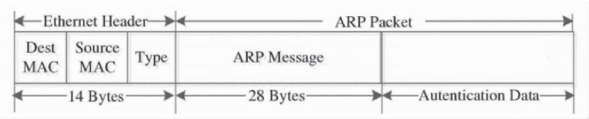

图 4 带有 payload 域的 ARP 消息

Fig. 4 ARP message with payload field

步骤 3:控制器为终端机建立验证信息表。控制器收到终端机的公钥和其它信息之后,在该终端机的交换机 ID 表项中创建一个验证信息表,如图 5 所示,包括交换机身份 ID、终端机的 IP 地址和 MAC 地址、初始接入端口、终端机的公钥和为该终端机分配的密钥对中的私钥。

交换机ID	IP	MAC
交换机端口号	主机公钥	控制器私钥
预留区域		

图 5 终端机验证信息表项

Fig. 5 Terminal verification information table entry

步骤 4:控制器给每个终端机分配特征值并生成 RSA 公钥和私钥。控制器先以自身特征值随机生成一对公钥和私钥,将私钥存入对应终端机的验证信息表项中。

步骤 5:控制器向终端机分发 RSA 公钥。控制器将生成的密钥对中的公钥封装在 ARP 的 payload 域并发送给对应终端机,终端机收到后将此公钥存在本地,此步骤的目的是为了防止验证过程中中间人窃取验证的信息。

(2)攻击防御阶段主要分为 3 个步骤

步骤 1:验证源地址对应信息。当控制器收到 ARP 请求包时,依照源 MAC 地址找寻对应的表项,如果交换机身份 ID、MAC 地址和 IP 地址、连接端口都相同的话,则以 ARP Reply 消息回复此数据包;如果 MAC 地址和 IP 地址映射不正确,则要继续验证,如果验证无法通过,则将此报告给管理人员,类型是攻击者在网络层或链路层的 ARP 攻击;如果 MAC 地址和 IP 地址的映射正确,但是与交换机的连接端口不正确,则需要进一步验证;如果找不到该对应的表项,说明该终端机未向控制器注册信息,需要注册后才能通信。

当 MAC 地址和 IP 地址映射不正确时,首先将此次 ARP 请求的目的 IP 地址记录在对应的表项预留处,之后 DHCP 服务器请求该 MAC 地址最新对应的 IP 地址,并修改验证表项,然后再与此次 ARP 请求报文的 IP - MAC 映射进行比对,如果匹配不成功,则向管理员上报 ARP 攻击,类型为攻击者在网络层或链路层的 ARP 攻击,并删去预留的 IP 地址;如果匹配成功,以表项中预留的 IP 地址和此 IP 地址对应的 MAC 地址为源地址向请求终端机回复 ARP Reply,同时也删去预留的 IP 地址。

步骤 2:查验源 MAC 地址是否合法。如果可以查到 MAC 地址对应的表项,则直接比对交换机身份 ID、MAC 地址、IP 地址和连接端口等信息即可验证身份。如果 MAC 地址和 IP 地址映射正确,但是接入端口不正确时,则随机地生成一条信息,并以终端机对应的 RSA 公钥对信息进行加密,发送给终端机,此时如果终端机能回复正确的消息,则更新终端机验证信息表中终端机的位置,并且以表项中预留的 IP 地址和此 IP 地址对应的 MAC 地址为源地址向请求终端机回复 ARP Reply,并删去预留的 IP 地址;如果源终端机不能回复正确的消息,或者等待超时后,直接删去预留的 IP 地址。则向管理人员上报 ARP 攻击,类型为攻击者同时利用网络层和链路层的 ARP 攻击。

步骤 3:验证目的终端机是否离开。当终端机通过验证后,控制器向目标终端机的 IP 地址发送一个带有验证消息的数据包,如果目的终端机能回复正确的消息,并且接入端口等信息没有变化,就说明目的终端机没有移动;如果目的终端机能通过验证,但是接入端口等消息已经发送了变化,则修改终端机验证信息表;如果目的终端机不能通过验证,说明终端机已经离开,则等待目的终端机重新注册。

2.3 方案的有效性

(1)对于“攻击者在网络层或链路层”的 ARP 攻击

这种 ARP 攻击的特性是 IP 地址和 MAC 地址的映射关系一定不真实。本方案中,如果终端机要发送 ARP 请求数据包,必须先在控制器处注册验证信息。在通信过程中,终端机发起 ARP 请求,控制器接收之后会将 ARP 数据包的源 IP 地址和 MAC 地址与注册时的映射比较,如果映射不正确,则向 DHCP 服务器申请最新映射并比较,如果源终端机无法通过控制器的验证,则被判定为攻击者在网络层或链路层的 ARP 攻击。

(2)对于“攻击者同时利用网络层和链路层”的 ARP 攻击

这种 ARP 攻击的特性是 ARP 数据包中的 MAC 地址与终端机本身的 MAC 地址不同。在本方案中,控制器通过交换机上传的 Packet_in 报文中终端机的连接端口与验证信息表中连接端口比较,然后向源终端机发送验证消息,如果其可以通过验证,则更新源终端机对应

MAC 地址的表项,如果没有通过验证,则被判定为攻击者,同时利用网络层和链路层的 ARP 攻击。

3　核心代码

3.1　RSA 加密部分

该部分代码在控制器安全模块和终端机的防御模块中都需要使用,用于生成属于自己的公钥与私钥,并发给对方。

(1)计算最大公约数

```
// 寻找 num1、num2 的最大公约数
// GCD(num1, num2) = GCD(num2, num1 Mod num2)
Integer:Gcd(Integer:num1, Integer:num2)
While(num2 ! = 0)
  //计算余数
  Integer:remainder = num1 Mod num2
//计算 GCD(num2,remainder)
  num1 = num2
  num2 = remainder
End While

//GCD(num1,0)为 num1
Return num1
End Gcd
```

(2)计算大整数大次幂并对大的整数取模

```
#超大整数超大次幂然后对超大的整数取模(bs ^ ept) mod n
def ep_md(bs, ept, n):
    bin_a = bin(ept)[2:][::-1]
    r = len(bin_a)
    bs_a = []
    pre_bs = bs
    bs_a.append(pre_bs)
    for _ in range (r - 1):
        next_bs = (pre_bs * pre_bs) % n
        bs_a.append(next_bs)
        pre_bs = next_bs
    a_w_b = __multi (bs_a, bin_a)
    return a_w_b % n

def __multi (a, bin_a):
    result = 1
    for index in range(len(a)):
        a = a[index]
        if not int(bin_a[index]):
            continue
        result * = a
    return result
```

在此步骤中,使用到了 Montgomery 算法,该算法最大作用的是使 RSA 2 个素数在数值特别大时也可以使用,由于 Montgomery 与本文讨论的方向不同,故此处不再详细介绍。

(3)公钥与私钥生成

```
def cre_key(p, q):
n = p * q
#计算与 n 互质的整数个数 欧拉函数
fi = (p - 1) * (q - 1)
#选取 e,一般选取 65537,此处为示例故取小数值
    e = 13
    a = e
    b = fi
r, x, y = ext_gcd(a, b)
#计算出的 x 不能是负数,如果是负数,说明 p、q、e 选取失败
    while x < 0:
        e = e + 1
        if gcd(e, fi) ! = 1:
            continue
        a = e
        b = fy
        r, x, y = ext_gcd(a, b)
    d = x
return (n, e), (n, d)
# 加密,m 是明文,c 为密文
def encrypt(m, pub_key):
    n = pub_key[0]
    e = pub_key[1]
    c = ep_md(m, e, n)
    return c
# 解密,c 是密文,m 是明文
def decrypt(c, self_key):
    n = self_key[0]
    d = self_key[1]
    m = ep_md(c, d, n)
    return m
```

3.2 控制器安全模块

控制器部分的安全模块就是本文所提的方案,终端机在进入网络获得IP地址之后,就向控制器注册信息,包括IP地址、MAC地址、终端机接入端口和以自己特征值生成的RSA公钥,控制器为每个终端机维护这一表项。当有终端机发起ARP请求时,交换机转发给控制器,控制器比对表项,如果IP地址和MAC地址映射不正确,或者IP地址和MAC地址映射正确但终端机接入端口号不正确,则需要验证,验证通过就代替目标终端机回复ARP应答报文,验证不通过就向管理员报告ARP攻击。而验证不通过时,如果是IP地址与MAC地址不匹配,则为攻击者在网络层或链路层的ARP攻击,如果IP地址与MAC地址匹配,但是与接入接口不匹配,则为攻击者同时利用网络层和链路层的ARP攻击。

(1)原POX控制器数据包处理代码

该部分为原POX控制器处理数据包过程,主要是创建数据包对象,过滤掉无法解析的数据包和链路层发现协议数据包,并且当交换机的dpid不在ARP表中,将该减缓及匹配的虚拟网关加入ARP表中。

```
def _handle_openflow_PacketIn(self,event):
1)创建数据包对象,
dpid = event.connection.dpid
inport = event.port
packet = event.parsed
2)过滤无法解析的数据包,
if not packet.parsed:
log.warning("%i %i ignoring unparsed packet", dpid, inport)
return
3)如果交换机的dpid不在ARP表中,则为该交换机创建一个ARP表项,并将交换机匹配的虚拟网关加入ARP表中,
if dpid not in self.arpTable:
self.arpTable[dpid] = {}
for fake in self.fakeways:
  self.arpTable[dpid][IPAddr(fake)] = Entry(of.OFPP_NONE,dpid_to_mac(dpid))
4)过滤LLDP报文,
if packet.type == ethernet.LLDP_TYPE:
Return
(2)原POX控制器处理IPv4数据包
if isinstance(packet.next, ipv4):
log.debug("%i %i IP %s => %s", dpid, inport, packet.next.srcip, packet.next.dstip)
#尝试将缓冲区等待的数据包发出去
self._send_lost_buffers(dpid, packet.next.srcip, packet.src, inport)
1)如果包源IP在ARP表中,学习或者更新端口或MAC表项,并且过滤ARP表项的入端口和包源地址不匹配的情况,
if packet.next.srcip in self.arpTable[dpid]:
  if self.arpTable[dpid][packet.next.srcip] != (inport, packet.src):
   #向管理员报arp攻击
   print 'there is a arp attrack'
2)如果包源IP不在arp表中,对应包dpid和包源地址的条例创建为此包的匹配项目,
else:
  log.debug("%i %i learned %s", dpid, inport, packet.next.srcip)
  self.arpTable[dpid][packet.next.srcip] = Entry(inport, packet.src)
#尝试转发,取包的目的地址
dstaddr = packet.next.dstip
3)如果目的地址在ARP表中,有端口信息并将消息发出,
if dstaddr in self.arpTable[dpid]:
  #取表中目的交换机和目的地址对应的端口
  prt = self.arpTable[dpid][dstaddr].port
  #取表中目的交换机和目的地址对应的地址
  mac = self.arpTable[dpid][dstaddr].mac
  #排除出端口等于入端口的情况
  if prt == inport:
   log.warning("%i %i not sending packet for %s back out of the ""input port" % (dpid, inport, dstaddr))
  else:
   log.debug("%i %i installing flow for %s => %s out port %i"% (dpid, inport, packet.next.srcip, dstaddr, prt))
   #建立行为列表
   actions = []
   #添加动作,为设置目的MAC地址
   actions.append(of.ofp_action_dl_addr.set_dst(mac))
   #添加动作,为交换机发消息的出端口为ARP表中目的交换机的端口
```

```
        actions. append( of. ofp_action_output ( port = prt) )
        if self. wide:#如果为广泛匹配
        match = of. ofp_match( dl_type = packet. type, nw_
dst = dstaddr)
        else:
        match = of. ofp_match. from_packet( packet, in-
port)
    #添加新的流表项
        msg = of. ofp_flow_mod( command  = of. OFPFC_
ADD,
          #在 FLOW_IDLE_TIMEOUT 时间内,如果没有
报文触发,则该规则删除
          idle_timeout = FLOW_IDLE_TIMEOUT,
          #到达 OFP_FLOW_PERMANENT 时间时,无论
如何都删除该规则
          hard_timeout =  of. OFP_FLOW_PERMANENT,
          #设计数据包存储在数据路径中的缓冲区的 ID
          buffer_id = event. ofp. buffer_id,
          actions = actions,
          match = match)
        event. connection. send( msg. pack( ) )
    4)当目的地址不在 ARP 表中的情况,
    elif self. arp_for_unknowns:
      #如果交换机的 dpid 和目的地址不在缓冲区内,
创建该交换机的表项
        if ( dpid, dstaddr) not in self. lost_buffers:
         self. lost_buffers[ ( dpid, dstaddr) ] = [ ]
        #取缓冲区的表项
        bucket = self. lost_buffers[ ( dpid,dstaddr) ]
        #建立条例,当前的时间,条例最大存在时间,数
据包存储在数据路径中的缓冲区的 ID,入端口
        entry = ( time. time( ) + MAX_BUFFER_TIME, e-
vent. ofp. buffer_id, inport)
        bucket. append( entry)
        #如果 bucket 项目的长度大于交换机上用于未知
ip 的最大数据包数
        while len( bucket) > MAX_BUFFERED_PER_IP:
del bucket [0]
        # Expire things from our outstanding ARP list. . .
        self. outstanding_arps = { k: v for k, v in self. out-
standing_arps. iteritems( ) if v > time. time( ) }
        if ( dpid, dstaddr) in self. outstanding_arps:
        return
        # 目的地址表项过期时间为 4 s,不再接收
        self. outstanding_arps[ ( dpid, dstaddr) ] = time.
time( ) + 4
        #构建一个 ARP 消息
        r = arp( )
        #ARP 消息类型为以太网
        r. hwtype = r. HW_TYPE_ETHERNET
        #报文 proto 类型为 IP
        r. prototype = r. PROTO_TYPE_IP
        r. hwlen = 6
        r. protolen = r. protolen
        r. opcode = r. REQUEST
        r. hwdst = ETHER_BROADCAST
        #协议目的为目标地址
        r. protodst = dstaddr
        #源地址是请求包的地址
        r. hwsrc = packet. src
        #协议源是请求包源 IP
        r. protosrc = packet. next. srcip
        r. payload = '1111'
        e = ethernet ( type = ethernet. ARP_TYPE, src =
packet. src,dst = ETHER_BROADCAST)
        #将 ARP 消息封装在链路层协议中
        e. set_payload( r)
        log. debug( "%i %i ARPing for %s on behalf of %
s" % ( dpid, inport,r. protodst, r. protosrc) )
        #指示交换机发送数据包
        msg = of. ofp_packet_out( )
        #将 e 封装在 msg 的数据域中
        msg. data = e. pack( )
        msg. actions. append ( of. ofp_action_output( port =
of. OFPP_FLOOD) )
        msg. in_port = inport
        event. connection. send( msg)
    (3)处理 ARP 数据包的 ARP 防御模块
    elif isinstance( packet. next, arp) :
    1)创建一个 ARP 数据包对象,并过滤一些数据包,
    a = packet. next
    log. debug( "%i %i ARP %s %s  = > %s", dpid,
inport, { arp. REQUEST: "request", arp. REPLY: "reply"} .
get( a. opcode,'op:%i' % ( a. opcode,) ) , a. protosrc, a.
protodst)
    #过滤出 ARP 数据包
```

```
if a.prototype == arp.PROTO_TYPE_IP:
  #如果是链路层协议
  if a.hwtype == arp.HW_TYPE_ETHERNET:
   #并且包的协议源地址不为0.0.0.0
   if a.protosrc != 0:
    #过滤目的地址为控制器本身的数据包
if a.protodst == '10.168.1.1':
```

2)当ARP数据包为密钥交换数据包时,为终端机分发私钥并记录终端机的公钥,

```
          if a.payload != '':
            if '(' in str(a.payload):
             #为该终端机生成密钥
             pubkey, selfkey = gen_key(p, q)
             self.macTable[packet.src] = {}
             #将a负载域中的密钥记录在对应的表项中
             self.macTable[packet.src]['host_pubkey'] = a.payload
             #将生成的密钥加入到a对应的表项中
              self.macTable[packet.src]['controller_prakey'] = selfkey
             r = arp()
             r.hwtype = a.hwtype
             r.prototype = a.prototype
             r.hwlen = a.hwlen
             r.protolen = a.protolen
             r.opcode = arp.REPLY
             r.hwdst = a.hwsrc
             r.protodst = a.protosrc
             r.protosrc = a.protodst
             r.payload = str(pubkey)
             r.hwsrc = a.hwdst
             e = ethernet(type = packet.type, src = dpid_to_mac(dpid),dst = a.hwsrc)
             e.set_payload(r)
             log.debug("%i %i answering ARP for %s" % (dpid, inport,r.protosrc))
             msg = of.ofp_packet_out()
             msg.data = e.pack()
              msg.actions.append(of.ofp_action_output(port = of.OFPP_IN_PORT))
             msg.in_port = inport
             event.connection.send(msg)
             return
```

3)当ARP数据包为验证信息数据包时,用终端机的公钥解密,并比对控制器本身生成的验证信息,

```
          else:
           recheck_prakey = self.macTable[packet.src]['controller_praker']
           recheck_msg = a.payload
           #用a的公钥解密
            recheck_num = decrypt(recheck_msg, recheck_prakey)
           #比对解密后的信息与自己分发的信息
           if recheck_num == self.macTable[packet.src]['check_num']:
             self.arpTable[dpid][a.protosrc].port = inport
              if self.macTable[packet.src]['request_ipaddr'] in self.arpTable[dpid]:
              # We have an answer...
              # if not self.arpTable[dpid][a.protodst].isExpired():
              r = arp()
              r.hwtype = a.hwtype
              r.prototype = a.prototype
              r.hwlen = a.hwlen
              r.protolen = a.protolen
              r.opcode = arp.REPLY
              r.hwdst = a.hwsrc
              r.protodst = a.protosrc
               r.protosrc = self.macTable[packet.src]['request_ipaddr']
              r.hwsrc = self.macTable[packet.src]['request_macaddr']
              e = ethernet(type = packet.type, src = dpid_to_mac(dpid),dst = a.hwsrc)
              e.set_payload(r)
              log.debug("%i %i answering ARP for %s" % (dpid, inport,r.protosrc))
              msg = of.ofp_packet_out()
              msg.data = e.pack()
              msg.actions.append(of.ofp_action_output(port = of.OFPP_IN_PORT))
                msg.in_port = inport.connection.send(msg)
              return
```

```
                else:
    #上报管理员发生网络层和链路层的攻击
                    print 'there is a arp attrack which steal and or mac'
                    return
                else:
                    return
```

4)如果终端机的协议 IP 地址存在于主机验证表项,则比对与 MAC 地址和接入端口的对应关系,判断是否发生 ARP 攻击,

```
    if a. protosrc in self. arpTable[dpid]:
    #a 的 IP 地址与 MAC 地址不对应,则上报管理员有网络层或链路层的 ARP 攻击
            if self. arpTable[dpid][a. protosrc]. mac ! = packet. src:
                #上报管理员发生网络层或链路层的攻击
                print 'there is a arp attrack which only steal ip or mac'
                return
            else:
                #如果 a 的 IP 地址和 MAC 地址都与验证表中相同,但输入端口不相同
                if self. arpTable[dpid][a. protosrc]. port ! = inport:
                    self. macTable[packet. src] ['request_ipaddr'] = a. protodst
                    self. macTable[packet. src]['request_macaddr'] = packet. src
                    check_pubkey = self. macTable[packet. src]['host_pubkey']
                    check_num = random. randint(0, 300)
                    check_msg = encrypt(check_num, check_pubkey)
                    self. macTable[packet. src]['check_num'] = str(check_msg)
                    r = arp()
                    r. hwtype = a. hwtype
                    r. prototype = a. prototype
                    r. hwlen = a. hwlen
                    r. protolen = a. protolen
                    r. opcode = arp. REPLY
                    r. hwdst = a. hwsrc
                    r. protodst = a. protosrc
                    r. protosrc = a. protodst
                    r. payload = str(check_msg)
                    r. hwsrc = self. arpTable[dpid][a. protodst]. mac
                    e = ethernet(type = packet. type, src = dpid_to_mac(dpid),dst = a. hwsrc)
                    e. set_payload(r)
                    log. debug("%i %i answering ARP for %s" % (dpid, inport,r. protosrc))
                    msg = of. ofp_packet_out()
                    msg. data = e. pack()
                    msg. actions. append(of. ofp_action_output(port = of. OFPP_IN_PORT))
                    msg. in_port = inport
                    event. connection. send(msg)
                    #上报管理员发生网络层和链路层的攻击
                    print 'there is a arp attrack which steal and or mac'
                    return
                else:
                    log. debug("%i %i learned %s", dpid, inport, a. protosrc)
                    self. arpTable[dpid][a. protosrc] = Entry(inport, packet. src)
                    self. _send_lost_buffers(dpid, a. protosrc, packet. src, inport)
    if a. protosrc in self. arpTable[dpid]:
```

5)如果终端机身份验证成功,则正常回复 ARP 数据包,

```
            if self. arpTable[dpid][a. protosrc] = = (inport, packet. src):
                if a. opcode = = arp. REQUEST:
                # Maybe we can answer
                if a. protodst in self. arpTable[dpid]:
                    if not self. arpTable[dpid] [a. protodst]. isExpired():
                        r = arp()
                        r. hwtype = a. hwtype
                        r. prototype = a. prototype
                        r. hwlen = a. hwlen
                        r. protolen = a. protolen
                        r. opcode = arp. REPLY
                        r. hwdst = a. hwsrc
```

```
r. protodst = a. protosrc
r. protosrc = a. protodst
r. payload = '111111'
 r. hwsrc = self. arpTable[dpid][a. protodst]. mac
e = ethernet(type = packet. type, src = dpid_to_mac(dpid), dst = a. hwsrc)
e. set_payload(r)
log. debug("%i %i answering ARP for %s" % (dpid, inport, r. protosrc))
msg = of. ofp_packet_out()
msg. data = e. pack()
msg. actions. append (of. ofp_action_output (port = of. OFPP_IN_PORT))
msg. in_port = inport
event. connection. send(msg)
return
log. debug("%i %i flooding ARP %s %s = > %s" % (dpid, inport, {arp. REQUEST: "request", arp. REPLY: "reply"}. get(a. opcode, 'op:%i' % (a. opcode,)), a. protosrc, a. protodst))
msg = of. ofp_packet_out(in_port = inport, data = event. ofp, action = of. ofp_action_output(port = of. OFPP_FLOOD))
event. connection. send(msg)
```

3.3 终端机安全模块

终端机安全模块主要包括RSA加密部分和以下用于信息验证部分,信息验证部分包括了ARP嗅探,建立与控制器匹配的密钥表项和发送验证ARP数据包。

```
f_arp_msg = sniff(filter = 'ARP')
if f_arp_msg. load is not None:
    if '(' in f_arp_msg. load:
        macTable['mac'] = f_arp_msg. hwsrc
        macTable['mac']['pubkey'] = f_arp_msg. load
    else:
        pkt_hwsrc = f_arp_msg. hwsrc
        pkt_psrc = f_arp_msg. psrc
        check_msg = random. randint(0,300)
          check _ pubkey = macTable ['mac'] ['pubkey']
        check _ im = encrypt (check _ msg, check _ pubkey)
```

4 方案示例与测试

4.1 测试环境

实验的仿真模拟环境中,需要一台虚拟机,具体配置如下:

(1)操作系统为Linux Ubuntu14.04 LTS或者更高版本;

(2)CPU英特尔i7处理器,主频2.5 GHz;

(3)内存2GB或者更高配置;

(4)运行SDN控制器的网络模拟平台。

其中,SDN控制器选用POX 0.5.0开源控制器,使用Open vSwitch交换机,虚拟网络运用Mininet 2.3.0网络仿真平台构建,南向接口协议使用OpenFlow v1.0协议,使用Wireshark 2.6.6作为流量分析工具,其次使用Scapy 2.4.3[36]作为ARP攻击的手段,编程语言为Python 2.7。

4.2 实验过程

实验建立如图6的网络拓扑。首先启动POX控制器,如图7,然后执行mn命令创建一个简单的网络拓扑,如图8,控制器指定为POX,使用Open vSwitch作为交换机。

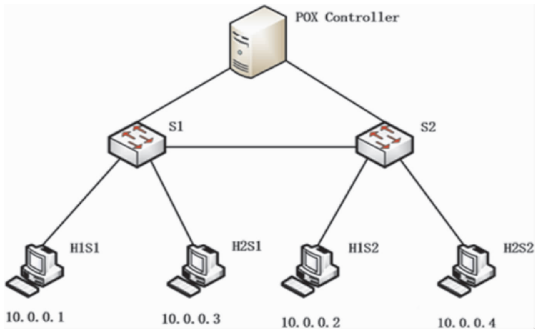

图6 本实验网络拓扑

Fig. 6 Network topology of this experiment

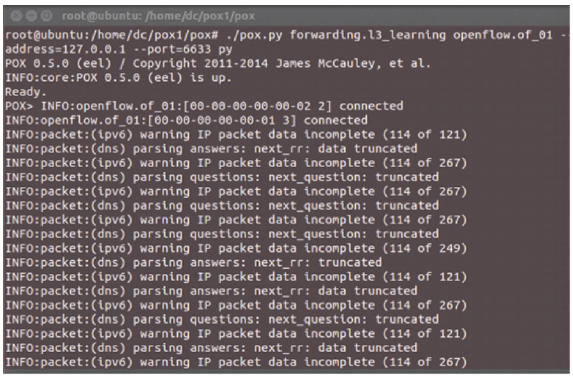

图7 启动pox控制器

Fig. 7 Start pox controller

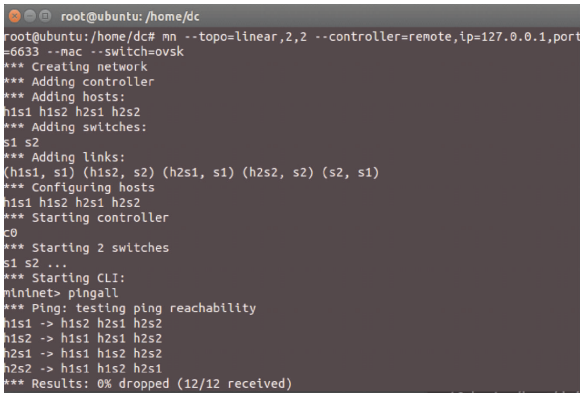

图 8　建立网络拓扑

Fig. 8　Establish network topology

假设图 6 中 H1S2 为攻击者,对 H2S2 发起攻击,攻击的类型分为 2 种:①针对网络层或链路层的 ARP 攻击;②针对网络层和链路层的 ARP 攻击。

(1)攻击者在网络层或链路层的 ARP 攻击

攻击者在网络层或链路层的 ARP 欺骗程序代码如图 9,欺骗程序先盗用 H1S1 的 IP 地址,同时每间隔 3 s 向终端机 H2S2 发送 ARP 请求报文。首先观察没有使用方案之前该网络遭受此类 ARP 攻击的情况。终端机 H1S2 发起攻击之后,交换机 S2 由于没有对应的流表项,于是把消息封装成 Packet_in 消息,上传给控制器,此时控制器已经受到了欺骗,控制器将该数据包的相关信息记录后,指示交换机 S2 将其洪泛,于是终端机 H2S2 接收到了终端机 H1S2 的数据包,同时修改自身的 ARP 缓存(图 10),并进行回复,交换机 S2 又因为没有找到对应流表项,于是上报控制器,控制器记录该数据包的信息后,向 S2 下发流规则,指示 S2 将该数据包转发给终端机 H1S2。之后终端机 H2S2 再与 H1S1 通信,终端机 H2S2 对 H1S1 发出的数据包都会发送给终端机 H1S2,所以 ARP 攻击完成。

```
from scapy.all import *
import time


a = scapy.all.ARP(psrc='10.0.0.1',pdst='10.0.0.4',
                  hwsrc='00:00:00:00:00:02',
                  hwdst = 'ff:ff:ff:ff:ff:ff')

while True:
    if time.time() % 3 ==0:
        send(a)
    else:continue
```

图 9　攻击者在网络层或链路层的 ARP 欺骗程序代码

Fig. 9　The attacker's ARP spoofing program code at the network or link layer

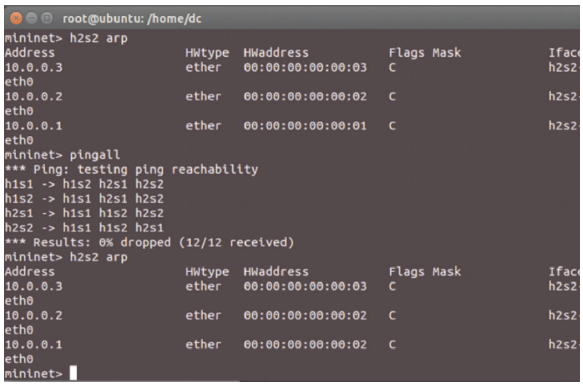

图 10　未使用方案前终端机 H2S2 受攻击后 ARP 表项

Fig. 10　ARP table entries after the terminal H2S2 is attacked before the scheme is not used

使用方案之后,4 台终端机都向控制器注册信息,密钥交换的过程是终端机向控制器发送自己的公钥,同时控制器向终端机发送自己的公钥,如图 11 和图 12。此时终端机 H1S2 再发起 ARP 攻击时,交换机 S2 因为没有流,将该数据包上报控制器,控制器比对终端机验证信息之后,发现映射不正确,于是向管理员报告 ARP 攻击的错误。并且这个时候终端机 H1S1 与终端机 H2S2 为正常通信,具体攻击结果如图 13 所示。使用方案之后终端机 H2S2 与终端机 H1S1 通信结果如图 14 所示。

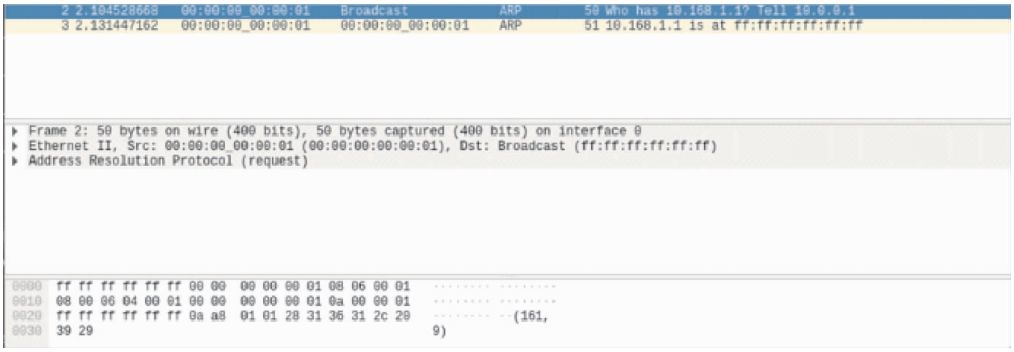

图 11　终端机向控制器发送自己的公钥

Fig. 11　The terminal sends its own public key to the controller

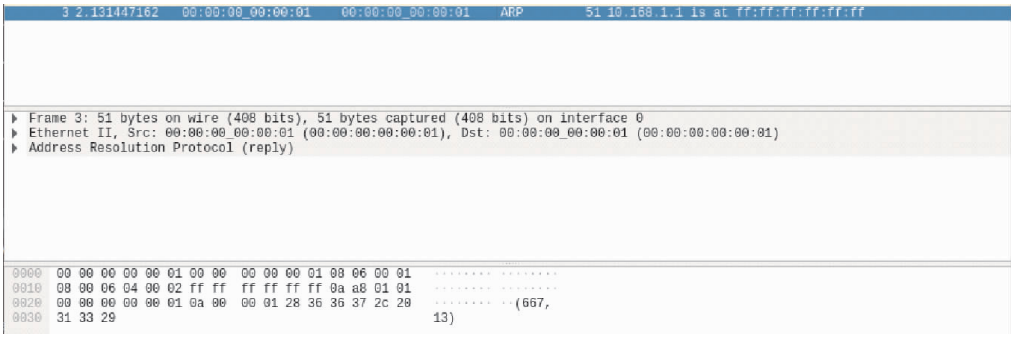

图 12 控制器向终端机发送自己的公钥

Fig. 12 The controller sends its own public key to the terminal

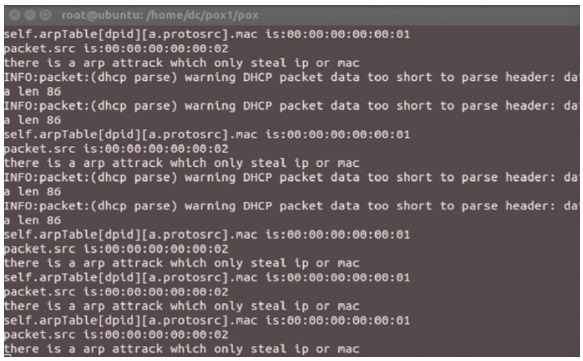

图 13 攻击者在网络层或链路层的 ARP 攻击结果

Fig. 13 The attacker's ARP attack result at the network or link layer

(2)攻击者同时利用网络层和链路层的 ARP 攻击

攻击者同时利用网络层和链路层的 ARP 攻击程序代码如图 15 所示。4 台终端机都向控制器注册了信息,当终端机 H1S2 开启攻击程序之后,向网络中传输 ARP 欺骗包,S2 在收到欺骗数据包后,由于找不到对应的流表项,于是封装消息后,将消息数据包发送到控制器,控制器比对验证信息表之后,发现 IP、MAC 地址与输入端口的映射与表项中的信息不同,于是向终端机 H1S2 发送验证信息,因为终端机 H1S2 无法回答正确的消息,控制器将向管理员上报系统发生了 ARP 攻击,攻击类型是攻击者同时利用网络层和链路层的 ARP 攻击,攻击结果如图 16 所示。

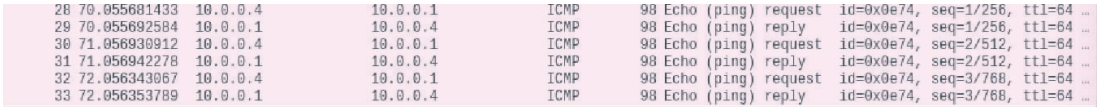

图 14 使用方案之后终端机 H2S2 与终端机 H1S1 通信

Fig. 14 After using the plan, the terminal H2S2 communicates with the terminal H1S1

```
from scapy.all import *
import time

a = scapy.all.ARP(psrc='10.0.0.1',pdst='10.168.1.1',
                  hwsrc='00:00:00:00:00:01',
                  hwdst = 'ff:ff:ff:ff:ff:ff')

while 1:
    if time.time() % 2 == 0:
        send(a)
    else:continue
```

图 15 攻击者同时利用网络层和链路层的 ARP 攻击代码

Fig. 15 The attacker uses both the network layer and link layer ARP attack code

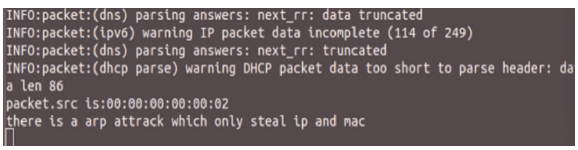

图 16 攻击者同时利用网络层和链路层的 ARP 攻击结果

Fig. 16 The attacker uses both the network layer and the link layer's ARP attack results

5 结束语

文章分析了现有网络架构存在的问题,概括了软件定义网络所具有的特点,在分析了 SDN 架构下 ARP 攻击原理和形式之后,提出了解决 SDN 架构下 ARP 攻击的方案。但是本文所提方案验证中终端机验证信息表比较大,在控制器中占用的空间也比较大,而且在终端机下线之后也必须维持这样的表项,当接入的终端机变多的时候控制器的存储空间会变得紧缺,因此,在以后的方案改进中可以考虑更小的表项,并且能动态地存储表项。还有验证 IP 地址与 MAC 地址映射时使用了 DHCP 服务器,而控制器与 DHCP 服务器之间的通信安全没有保障,这是下一步的完善方向。

参考文献:

[1] 黄韬,刘江,魏亮,等. 软件定义网络核心原理与应用实践[M]. 北京:人民邮电出版社,2014.

[2] 孔祥彬. 软件定义联网的路由选择技术研究[D]. 南京:南京邮电大学,2017.

[3] 严敏瑞. 软件定义网络中的 ARP 攻击解决方法研究[D]. 西安:西安电子科技大学,2017.

[4] Bruschi D,Ornaghi A,Rosti E. S-ARP: A secure address resolution protocol[C]//Computer Security Applications Conference, 2003. Proceedings. 19th Annual. Piscataway:IEEE,2004:66-67.

[5] Lootah W,Enck W, McDaniel P. TARP:Ticket-based address resolution protocol[J]. Computer Networks,2007,51(15):4322-4337.

[6] Limmaneewichid P, Lilakiatsakun W. P-ARP: A novel enhanced authentication scheme for securing ARP[C]//Proceeding of International Conference on Computer Communication and Management (ICCCM 2011). 成都: 亚昂教育咨询有限公司,2011:92-96.

[7] Hong S, Xu L, Wang H, et al. Poisoning network visibility in software-defined networks:New attacks and countermeasures[C]//NDSS Symposium. 2015,doi:10.14722/ndss.2.15.23283.

[8] 陈运,龚耀寰. 大数幂剩余的二进制冗余数 Montgomery 算法[J]. 电子科技大学学报,2000,29(6):587-590.

[9] Wei J, Fang B X, Zhang H L, et al. Optimal network security strengthening using attack-defense game model[C]//Sixth International Conference on Information Technology: New Generations. Piscataway: IEEE, 2009:475-480.

[10] McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: Enabling innovation in campus networks[J]. ACM SIGCOMM Computer Communication Review,2008, 38(2): 69-74.

[11] Yang Y M, Mao D M. Analysis of OpenFlow in mobile Ad hoc network[J]. Information Technology,2016,5(28):109-111.

[12] 杨泽卫,李呈. 重构网络:SDN 架构与实现[M]. 北京:电子工业出版社,2017.

[13] 张纯彬. SDN 跨域带宽分配机制研究与实现[D]. 北京:北京邮电大学,2016.

[14] Othmane B, Mouad B M, Redouane B. An overview on SDN architectures with multiple controllers[J]. Journal of Computer Networks and Communications,2016(2):1-8.

[15] 罗源. 基于 SDN 流表的回溯式负载均衡技术的研究与实现[D]. 柳州:广西科技大学,2018.

[16] 曲福. 基于 SDN 的网络虚拟化的研究与应用[D]. 北京:北京邮电大学,2016.

[17] 袁晶,宋旭翊,孟赟. OpenFlow 流表性能优化研究[J]. 有线电视技术,2017,11(19):74-76.

[18] 王保平. 基于 Neutron 的流量瓶颈及其可控性关键问题研究[D]. 北京:北京工业大学,2015.

[19] 胡芬. SDN 流量智能监测技术研究[D]. 长沙:湖南大学,2017.

[20] Fu Y, Fang L, Meng F, et al. An intelligent network attack detection method based on RNN[C]//2018 IEEE Third International Conference on Data Science in Cyberspace (DSC). Piscataway: IEEE, 2018,DOI:10.1109/DSC.2018.00078.

[21] 李洁. 云平台 SDN 关键技术的研究与展望[J]. 软件,2015,7(15):77-80.

[22] 桂兴亮. 基于 OpenFlow 的 SDN 控制器研究与设计[D]. 镇江:江苏科技大学,2016.

[23] Shafiq M, Tian Z, Bashir A K, et al. Data mining and machine learning methods for sustainable smart cities traffic classification: A survey[J]. Sustainable Cities and Society, 2020, DOI:10.1016/j.scs.2020.102177.

[24] 姜立立,曾国荪,丁春玲. 数据流特征感知的交换机流表智能更新方法[J]. 计算机应用,2016,36(7):1772-1778.

[25] 蔡明娟. 基于 OpenFlow 协议的网络虚拟化安全技术研究[D]. 武汉:华中科技大学,2014.

[26] 李民,陈向瑞,姚艳秋. 基于 ARP 协议的攻击分析与防范[J]. 长春师范学院学报(自然科学版),2010,29(1):41-44.

[27] 马钺. 基于 ARP 协议的网络攻防技术实践[J]. 数码世界,2018(1):323-325.

[28] 王榕国. ARP 欺骗网络安全研究[J]. 信息通信,2017(1):93-96.

[29] 刘晓静,王洪波. ARP 数据包详解[J]. 网络运维与管理, 2014(2):89-91.

[30] Wang Z H, Liu C G, Qiu J, et al. Automatically traceback RDP-based targeted ransomware attacks[J]. Wireless Communications and Mobile Computing, 2018,2018:1-13.

[31] 顾宪锋. 基于 OpenFlow 的虚拟网实现研究[J]. 通信技术, 2015,48(3):335-341.

【责任编辑: 孙向荣】