

基于 SDN 的一种隐藏端口检测方法

刘玉明¹, 王 勇^{1,2}, 俸 皓^{1,2}

(1 桂林电子科技大学 计算机与信息安全学院, 广西 桂林 541004;

2 桂林电子科技大学 广西云计算与复杂系统高校重点实验室 广西 桂林 541004)

摘 要: 考虑基于 SDN 架构的网络环境, 并针对当前检测隐藏端口的方法的不足, 提出一种全新的基于 SDN 架构的隐藏端口检测方法. 利用 SDN 集中控制的特性, 通过内存映射流表项的方法来实时提取主机的连接信息, 并结合主机代理的信息进行交叉视图检测, 同时在检测过程中引入检测状态机, 使得准确检测出部署环境下所有主机的隐藏端口. 实验结果表明, 该方法能高效地检测主机恶意程序隐藏的端口, 并具有良好的兼容性和系统性能.

关键词: 隐藏端口; 木马; SDN; 交叉视图检测

中图分类号: TP391

文献标识码: A

文章编号: 1000—7180(2017)10—0135—06

A Method of Detecting Hidden Ports Based on SDN

LIU Yu-ming¹, WANG Yong^{1,2}, FENG Hao^{1,2}

(1 School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin 541004, China; 2 Guangxi Colleges and Universities Key Laboratory of Cloud Computing and Complex Systems, Guilin University of Electronic Technology, Guilin 541004, China)

Abstract: This paper proposes a method of hidden-ports detection based on SDN, with the characteristics of controlling the whole network, the controller could retrieve all the sessions' information, combined with the customized data received from proxy, the controller could find all the sessions from hidden-ports based on the cross-view of full information and visible information. As the experiment shows, this method could detect all the hidden-ports effectively and compatibly.

Key words: hidden ports; trojan; SDN; cross-view

1 引言

当前网络环境对网络安全状况的监控大多通过部署入侵检测系统来检测攻击流量, 并联合部署的防护设备阻截攻击的流量. 随着网络服务的不断丰富和网络环境的日益复杂, 面对各种各样的 APT (Advanced Persistent Threat) 攻击技术^[1], 传统网络设备很难全面检测并一一阻截, 在主机被渗透并被植入隐藏端口的木马或后门的情况下, 通过现有工具难以发现其传输消息的端口, 同时木马的通信流量在上行特征不明显的情况下也难以被监控设备

所识别^[2]. 因此, 在主机检测和网络检测都判断正常的情况下, 木马的生存时间将大大延长, 这意味着云主机租户的损失也越大. 本文研究重点在于, 如何通过实时监控快速发现隐藏端口的通信, 并对该异常连接采取措施, 使之不影响主机正常业务的进行.

为了解决传统网络管理和配置复杂的问题, SDN 作为一种新的网络架构被提出, 其核心思想在于传统网络设备的数据平面与控制平面的分离, 数据平面仅按照上层提供的规则处理数据的转发, 而控制平面则负责制定数据转发规则, 以及注册事件来提取和处理网络数据, 从而通过编程定义控制逻

收稿日期: 2017—01—03; **修回日期:** 2017—02—20

基金项目: 国家自然科学基金项目(61172053, 61163058, 61662018); 广西自然科学基金(2016GXNSFAA380153); 广西云计算与复杂系统高校重点实验室资助项目(2015209)

辑,即可完成对整个网络的集中式控制.该架构使得平面之间具有较低的耦合性,从而有利于网络的测试、管理、扩展.基于上述优点,SDN 被越来越多地部署到云中心,而 SDN 具有的固有优势,也为其在网络安全领域的应用提供了新的研究思路.

本文分析了端口隐藏的诸多方法,考虑基于 SDN 架构的网络环境,并针对当前检测机制在系统兼容性和检测效果方面的不足,提出了基于 SDN 的隐藏端口检测方法,并设计实现系统 SDHPDS(Software Defined Hidden Port Detection System).

2 相关工作

针对隐藏端口实现的研究,Zander S,Armitage G 等人系统地描述了基于网络协议隐写传输信息的方法,并给出了相应的对抗策略^[3],然而通过隐写的隐藏手段并不能躲避本地抓包工具的分析.

在对隐藏端口检测的研究中,已知文献多将其作为 Rootkit 隐藏对象检测的一部分来进行研究.李博等人对当前的操作系统隐藏对象检测机制进行了比较科学的分类,即按照实现和部署方式分为应用层检测、操作系统级检测、硬件辅助检测和虚拟化层检测.同时作者阐述了应用层检测和操作系统级检测的不可靠性,以及硬件辅助检测的高代价和不易兼容的特性,并实现基于高权限的 VMM 的隐藏对象检测系统 vDetector^[4].然而针对隐藏端口的检测而言,vDetector 在寻找可信连接时并不适用于 Windows 系统.

在基于 SDN 的安全领域研究方面,根据文献^[5]的阐述,目前针对 SDN 在网络安全方面的研究大致归纳为三个方面:一是 OpenFlow 协议本身的安全性研究;二是基于 SDN 架构的环境安全性的研究;三是 SDN 在网络安全领域的应用研究.这与 Farhady H 等人在总结 SDN 安全相关的内容时,使用数据平面、控制平面、应用平面来归纳的思路类似^[6].然而目前 SDN 在网络安全领域的研究多涉及异常流量的检测和防御,仅有少量文献研究利用 SDN 的特性来检测隐藏对象,相关性较高的仅有文献^[7],其结合入侵检测、虚拟机内视技术以及 SDN 达到检测云主机隐藏攻击进程的目的,虽然该系统仅针对进行网络攻击的隐藏进程实体进行检测,但该文文献为隐藏对象检测提供了新的思路.

3 系统设计

3.1 工作场景

SDHPDS 的实现基于 SDN 的网络架构,在不

考虑网络监控和防护设备的前提下,图 1 给出简化后的 SDHPDS 工作场景图. SDHPDS 架构下的主机根据自身的系统类型引入相应的代理服务,并通过自定义分组接入不同的 SDN 交换机,同时 SDN 交换机与一台控制器进行交互,因而对控制器功能进行扩展即可定制对整个架构下的网络连接和流量信息进行捕获和分析.

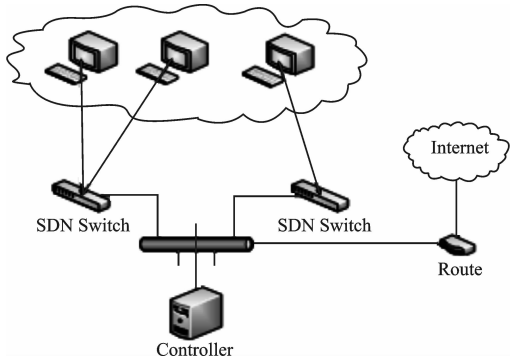


图 1 应用场景

3.2 系统总体框架

SDHPDS 旨在快速有效地检测其架构下所有主机的隐藏端口,其由代理服务、控制器核心、包过滤器、消息预处理器、隐藏端口检测器等组成,图 2 为 SDHPDS 的系统框架图.

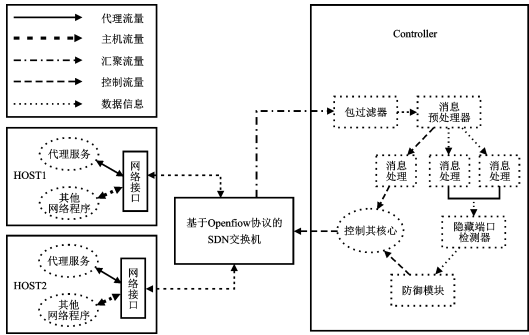


图 2 SDHPDS 系统架构

代理服务位于主机中,通过应用层接口获取获取用户可见的本地主机连接.通过应用层接口获取数据能够保持系统的兼容性,且易于系统功能的扩展.

控制器作为核心逻辑部件,由控制器核心、包过滤器、消息预处理器、隐藏端口检测器等组成.其中包过滤器负责接收来自 SDN 交换机的信息,并提取出消息给消息预处理器.消息预处理器首先对消息进行验证,然后调用相应的消息处理模块对消息进行处理.隐藏端口检测器在定时器触发或等待固定时间后,周期性地对数据进行交叉检测,一旦检测完

毕并识别出隐藏端口发起的连接,将触发防御处理,给控制器核心发送控制信息,禁止隐藏端口的通信.

3.3 系统结构设计

3.3.1 代理服务

代理服务周期性获取用户通过终端可见的网络连接信息,其工作模式为惰性模式,即发现建立新连接或断开旧连接后才会生成对应的代理信息,同时代理对多个生成的信息进行压缩,因此该方法在较大程度上能够减轻网络负载,代理服务工作模式如图 3 所示.

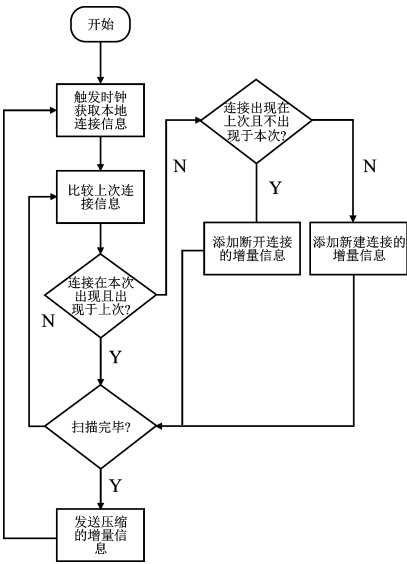


图 3 代理服务

隐藏端口的检测所需数据由主机代理服务和主机非代理程序生成,非代理程序的连接信息由 SDN 交换机捕获并由控制器记录,控制器在检测过程中不涉及其他模块对主机操作系统内核对象直接访问,这就使得 SDHPDS 具有良好的兼容性和平台扩展性.

3.3.2 包过滤器

包过滤器作为控制器的基本模块,完成 Open-flow 协议中对 PacketedIn 事件的响应,并在此基础上进行符合定制功能的过滤操作,包过滤器的工作流程如图 4 所示.

包过滤器实现包数据解析和解析后消息结构的提取,同时该架构的基本防护和过滤也应在这里实现.封包在经过 SDN 交换机时,判断是否发生 TableMiss,一旦发生则交付包过滤器进行处理.

包过滤器首先调用防护模块对数据包进行解析,对其安全性进行分析.非法数据包将被丢弃,而合法数据包则进入消息提取模块,提取模块工作完

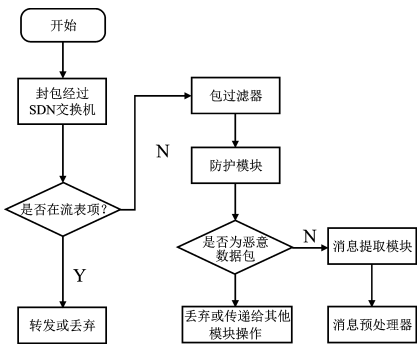


图 4 包过滤器

成后将消息进行封装,并交付消息预处理器进行下一步操作.值得一提的是,自动化的测试功能也可以通过定义控制指令,并扩展包过滤器来实现.

3.3.3 消息预处理器

消息预处理器为控制器的功能扩展模块.通过包过滤器提交的消息结构进行消息匹配,匹配完毕后触发相应的处理模块进行处理,其工作流程如图 5 所示.

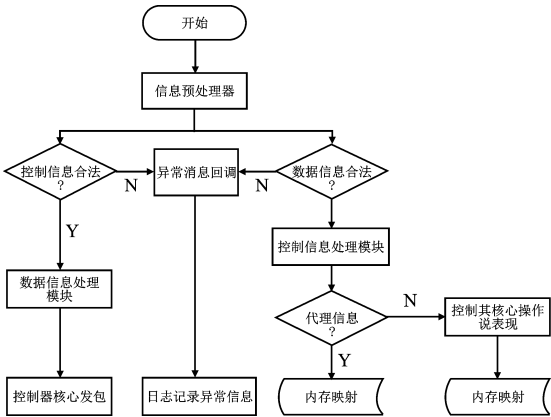


图 5 消息预处理器

如果匹配为控制消息且验证合法,则经过控制器核心进行相应的处理后,经由交换机回馈应答信息,否则触发异常处理模块,对异常的控制信息进行记录.

如果匹配为数据消息,同样在验证不合法情况下需进行记录.若验证合法还需判别是否为代理消息,若为代理信息,则需要内存中映射代理信息所表示的主机管理员可见的主机连接,否则将消息结构中包含的网络连接信息交付于控制器核心,用以建立流表项,并随后在内存中映射流表项.

值得注意的是之所以需要通过内存来映射流表项,是由于该系统的防御策略是针对某个连接进行的,为此需要在流表项中填充 IP 和 PORT 等信息,

而这很容易导致大量的流表项被占用甚至造成流表溢出,因此必须根据流表字段 `idle_timeout` 的定义来删除规定时间内不活动的连接表项. 然而如此操作会导致一个关键问题,即网络连接记录与流表项的记录不一致. 而为解决二者一致性问题,本文通过内存来映射流表项,来记录一段时间内的网络连接和代理信息.

3.3.4 隐藏端口检测器

为方便描述检测器检测隐藏端口的过程,以下给出网络连接实体的定义:

- (1) $e = (datapath_id, protocol, src_ip, src_port)$ 用以描述主机通信的一条连接;
- (2) $S_i = \{e \mid e \in \text{编号为 } i \text{ 的 SDN 交换机在检测周期内建立过的流表项}\}$;
- (3) $T_i = \{e \mid e \in \text{控制器对编号为 } i \text{ 的 SDN 交换机在检测周期内映射的流表项}\}$,显然有 $T_i = S_i$;
- (4) $V_k = \{e \mid e \in \text{编号为 } k \text{ 的主机在检测周期内通过代理获取管理员通过终端可见的连接}\}$;
- (5) $W_k = \{e \mid e \in \text{控制器对编号为 } k \text{ 的主机在检测周期内映射的代理信息}\}$,显然有 $W_k = V_k$;
- (6) $H = \{e \mid e \in \text{所有主机在检测周期内管理员不可见的连接}\}$;

无论采用哪一种端口隐藏的方法,其建立的连接都不会被管理员通过终端所发现,然而却会在经过 SDN 交换机处理的过程中被记录于流表项. 考虑在图 1 的应用场景下,假设 SDN 交换机与云主机部署数量分别为 m 和 n ,基于以上事实,隐藏端口建立连接的集合 H 满足如下关系:

$$H = \sum_{i=1}^m S_i - \sum_{k=1}^n V_k = \sum_{i=1}^m T_i - \sum_{k=1}^n W_k \quad (1)$$

式(1)表示隐藏连接集合.

然而在实际检测过程中,控制器获取网络连接产生的代理信息的时间要滞后于控制器直接获取网络连接信息的时间,即获取内存映射代理信息与内存映射网络连接的信息不同步,当判定发生时,有一部分正常连接被判定为隐藏连接. 为解决检测的实时性问题,引入检测状态机,如图 6 所示.

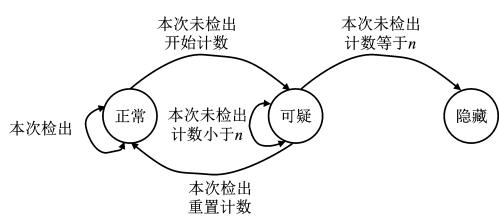


图 6 检测状态机

检测算法采用伪代码描述如下:

```
Detection state machine
Input: n, checking_period // 阈值和检测窗口
Output: hidden_entries{ }, suspicious_entries{ }
WHILE True:
    WaitForEvent(checking_period)
    AcquireLock()
    S = GetCurrentSet_S() // SDN 流表项全集 S
    V = GetCurrentSet_V() // 可见集合 V
    ReleaseLock()
    FOR e in S:
        IF e NOT in V:
            IF e NOT in suspicious_entries:
                suspicious_entries[e] = 0
            ELSE:
                suspicious_entries[e] += 1
                IF suspicious_entries[e] >= n:
                    hidden_entries[e] = 1
            ELIF e in suspicious_entries:
                DEL suspicious_entries[e]
    IF IsConditionToBreak():
        break
    yield((hidden_entries, suspicious_entries)) // 输出
    ResetEvent()
```

4 实验验证

测试环境由三台物理主机和 SDN 交换机组成. 部署为客户端的主机系统 Windows XP SP3, CPU 为 Intel Atom Z520(1.33 GHz), 内存 1 GB; 部署为服务端的主机系统 Windows7 SP1, CPU 为 Intel Core i7(3.6 GHz), 内存 8 GB; 部署为控制器的主机系统 Ubuntu14.04 LTS, CPU 为 Intel Core i7(3.6 GHz), 内存 8 GB; SDN 控制器为 Ryu4.3 版本.

4.1 系统测试

在该实验环境部署下,固定网络中的连接数为 500,并分别固定客户端和服务端的通信端口,以便每轮的测试环境保持一致. 在该 500 个连接中, Win7 服务器分配连接数为 10, XP 客户端分配连接数为 490,考虑到隐藏端口程序的共性,即对管理员不可见,因此选取劫持系统调用的驱动程序 HidePort_Hook 来其中的 51 个通信端口进行隐藏.

随着代理工作间隔(ProxyIntervalpi)和每个连接会话持续时间(SessionLastTimeslt)的变化,得到隐藏端口检测的误判率的结果如表 1 所示,同时得到漏判率全为 0,即无论参数如何变化,实验都已检

测出 51 个隐藏的端口. 接着,基于同样的实验环境,将会话连接设置 $>4\text{ s}$,代理间隔设置为 1.5 s ,引入阈值为 3,得到误判率和漏判率皆为 0.

表 1 系统误判率							
slt/s pi/s	1	1.5	2	2.5	3	3.5	4
1	49.6	37	53.4	53.2	58.2	57.6	61.0
1.5	6.4	3.8	44.4	42.8	58.8	57.4	65.8
2	0	0	21.4	22.6	44.6	44.0	56.2
2.5	0	0	0	0	30.2	28.0	43.8
3	0	0	0	0	14.6	15.0	32.8
3.5	0	0	0	0	0	1.2	22.8
4	0	0	0	0	0	0	10.2
>4	0	0	0	0	0	0	0

然后对系统的性能进行测试,在上述的测试环境下,客户端每条连接在会话生存时间内不间断收发包,此时以 10 s 为间隔计算控制器 CPU 负载的均值,并最终在会话时间结束后取均值的最大值. 同样该实验以代理间隔和会话时长作为影响因子,在极端取值的情况下,即分别取代理间隔,会话时长为 1 s 的情况,观察另一个影响因子对 CPU 负载的影响,结果如图 7 所示.

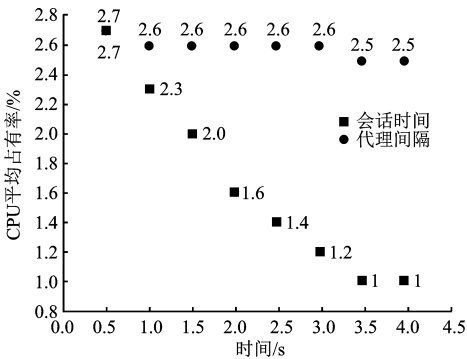


图 7 系统性能

4.2 实验分析

针对功能测试部分,由表 1 可知,当把一次检测中的可疑连接当做隐藏端口建立的连接处理,代理服务的工作间隔与连接的会话时间对误判率存在较大的影响,因为极端情况下连接会话时间过短或者 CPU 在调度程序的时候对代理服务分配时间片过少,都将可能导致连接在会话期间无法被代理服务记录,并且代理数据发向控制器也存在一定的时延. 然而无论连接会话时间多短,其都会被控制器第一时间记录,根据本文提出的隐藏端口检测方法可知,此时该连接会被判定为隐藏端口建立的连接,这就

造成了误判情况的发生. 这恰好说明了代理数据所反映的主机网络连接,与流表项中的记录不能实时匹配的问题,因此需要引入状态机来消除二者非实时匹配的特征.

考虑到隐藏端口的恶意程序一般会长期潜伏,并随开机启动并一直存在,则根据表 1 和检测状态机的处理方法可知,主机的隐藏端口一定能够被检测出来. 另外,即使主机的正常通信持续时间较短,如类似 Web 访问的短连接情况,实际上 Web 访问的 TCP 端口从结束访问的 TIME_WAIT 状态到彻底关闭连接的 CLOSED 状态,远超过了实验中的 4 s 预值,因此在状态机阈值控制范围内,除非挂起或者结束代理服务,否则难以出现误报的情况.

针对实验的性能测试部分,由图 7 可知,SDH-PDS 对系统影响较小,由于性能测试是在单位时间内取均值最大值,因此,即使对于长连接的压力测试,CPU 负载也不会有较大变化. 该在连接会话时长固定情况下,代理间隔越长,CPU 负载具有越小的趋势,由于代理对连接信息进行了有效压缩,控制器处理数据包的压力减小了,同时可发现代理间隔对 CPU 负载影响较小,显然这与代理的惰性工作模式有关,随着代理间隔时间增长,代理一次性压缩的数据就越多,其与控制交互的频率越低. 而在代理间隔固定的情况下,会话时长对 CPU 负载具有相对明显的影响,显然会话时间越长,控制器在两次流表操作和代理接收操作的时间间隔也越久,同样减轻了控制器 CPU 的负载,并且流表操作对于 CPU 而言具有相对明显的影响.

5 结束语

本文提出了一种基于 SDN 的隐藏端口检测方法,该方法利用了 SDN 对网络集中控制的特性,结合主机代理服务,能够快速发现 SDN 网络架构下隐藏端口的主机,而无论隐藏端口采用应用层、系统层、VMM 层、硬件层中哪种隐藏手段,同时该方法表现出良好的兼容性和系统性能. 然而该检测方法仅能证明主机隐藏端口的通信,却无法具体定位到主机上隐藏通信的特定模块,因此如何结合 SDN 控制器与代理各自的特点来定位主机隐藏通信的模块,将是下一步的研究内容.

参考文献:

[1] Jian-Fang L I, Zhang S D. Research on advanced persistent threat attack techniques[J]. Journal of Commu-

nication University of China, 2016.

[2] 彭国军, 王泰格, 邵玉如, 等. 基于网络流量特征的未知木马检测技术及其实现[J]. 信息安全, 2012, 12(10):5-9.

[3] Zander S, Armitage G, Branch P. Covert channels and countermeasures in computer network protocols[J]. IEEE Communications Magazine, 2007, 45(3):44-57.

[4] 李博, 沃天宇, 胡春明, 等. 基于 VMM 的操作系统隐藏对象关联检测技术[J]. 软件学报, 2013, 24(2): 405-420.

[5] 王蒙蒙, 刘建伟, 陈杰, 等. 软件定义网络:安全模型、机制及研究进展[J]. 软件学报, 2016, 27(4): 969-992.

[6] Farhady H, Lee H Y, Nakao A. Software-defined networking: A survey [J]. Computer Networks, 2015, 81(C):79-95.

[7] 崔竞松, 郭迟, 陈龙, 等. 创建软件定义网络中的进程级纵深防御体系结构[J]. 软件学报, 2014(10): 2251-2265.

作者简介:

刘玉明 男, (1991-), 硕士研究生. 研究方向为信息安全、云安全. E-mail: 367137144@qq. com.

王 勇 男, (1964-), 博士, 教授. 研究方向为云计算、计算机网络技术及应用、信息安全等.

俸 皓 男, (1978-), 博士研究生. 研究方向为计算机网络、无线传感器网络、软件定义网络及云计算等.

(上接第 134 页)

[9] 聂栋栋, 马勤勇. 新线路能量函数下基于线裁剪的图像自适应缩放[J]. 模式识别与人工智能, 2014, 27(11): 1040-1046.

[10] Avidan S, Shamir A. Seam carving for content-aware image resizing [J]. ACM Transactions on Graphics, 2007, 26(3):10-18.

作者简介:

张永梅 女, (1967-), 教授. 研究方向为图像处理、机器学习. E-mail: zhangym@ncut. edu. cn.

邢 阔 男, (1991-), 硕士研究生. 研究方向为图像处理与智能识别技术.

吴 攀 男, (1988-), 硕士研究生. 研究方向为图像处理.