Nguyen-Son Vo
Van-Phuc Hoang (Eds.)

334

Industrial Networks and Intelligent Systems

LNICST

6th EAI International Conference, INISCOM 2020
Hanoi, Vietnam, August 27–28, 2020
Proceedings

Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 334

More information about this series at http://www.springer.com/series/8197

Nguyen-Son Vo · Van-Phuc Hoang (Eds.)

Industrial Networks and Intelligent Systems

6th EAI International Conference, INISCOM 2020
Hanoi, Vietnam, August 27–28, 2020
Proceedings

Springer

Editors
Nguyen-Son Vo
Faculty of Electrical and Electronics Engineering
Duy Tan University
Da Nang, Vietnam

Van-Phuc Hoang
Le Quy Don Technical University
Hanoi, Vietnam

ISSN 1867-8211 ISSN 1867-822X (electronic)
Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering
ISBN 978-3-030-63082-9 ISBN 978-3-030-63083-6 (eBook)
https://doi.org/10.1007/978-3-030-63083-6

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

We are delighted to introduce the proceedings of the 2020 European Alliance for Innovation (EAI) International Conference on Industrial Networks and Intelligent Systems (INISCOM 2020). This conference has brought together researchers, developers, and practitioners from around the world who are leveraging and developing industrial networks and intelligent systems. The theme of INISCOM 2020 was "Computing, Telecommunications Technologies and Applications of 5G-IoT, AI and Cyber-Security to Improve Citizens' Lives."

The technical program of INISCOM 2020 consisted of 25 full papers in oral presentation sessions at the main conference tracks. The conference tracks were: Track 1 – Telecommunications Systems and Networks; Track 2 – Hardware, Software, and Application Designs; Track 3 – Information Processing and Data Analysis; Track 4 – Industrial Networks and Intelligent Systems; and Track 5 – Security and Privacy. Aside from the high-quality technical paper presentations, the technical program also featured one keynote speech. The keynote speaker was Prof. Dong-Seong Kim, from Kumoh National Institute of Technology, South Korea.

Coordination with the steering chairs, Prof. Imrich Chlamtac, Dr. Vien Ngo, and Dr. Ta Chi Hieu, was essential for the success of the conference. We sincerely appreciate their constant support and guidance. It was also a great pleasure to work with such an excellent Organizing Committee team and we thank them for their hard work in organizing and supporting the conference. In particular, the Technical Program Committee (TPC), led by our TPC co-chairs, Dr. Nguyen-Son Vo, Dr. Quoc Tuan Vien, and Prof. Trung Q. Duong, who completed the peer-review process of technical papers and made a high-quality technical program. We are also grateful to conference manager Natasha Onofrei for the support and all the authors who submitted their papers to INISCOM 2020.

We strongly believe that INISCOM provides a good forum for all researcher, developers, and practitioners to discuss all science and technology aspects that are relevant to industrial networks and intelligent systems. We also expect that the future INISCOM will be as successful and stimulating as indicated by the contributions presented in this volume.

October 2020

Nguyen-Son Vo
Van-Phuc Hoang

Organization

Steering Committee

Imrich Chlamtac	University of Trento, Italy
Vien Ngo	Queen's University Belfast, UK
Ta Chi Hieu	Le Quy Don Technical University, Vietnam

Organizing Committee

General Chair

Van-Phuc Hoang	Le Quy Don Technical University, Vietnam

General Co-chairs

Cong-Kha Pham	The University of Electro-Communications, Japan
Xuan-Nam Tran	Le Quy Don Technical University, Vietnam

TPC Chair and Co-chairs

Nguyen-Son Vo	Duy Tan University, Vietnam
Quoc Tuan Vien	Middlesex University, UK
Trung Q. Duong	Queen's University Belfast, UK

Sponsorship and Exhibit Chairs

Luong Duy Manh	Le Quy Don Technical University, Vietnam
Hoa Le-Minh	Northumbria University, UK

Local Chairs

Tran Cong Manh	Le Quy Don Technical University, Vietnam
Van-Trung Nguyen	Le Quy Don Technical University, Vietnam
Do Thanh Quan	Le Quy Don Technical University, Vietnam

Workshops Chairs

Koichiro Ishibashi	The University of Electro-Communications, Japan
Sylvain Guilley	Télécom Paris, France
Xuan-Tu Tran	VNU University of Engineering and Technology, Vietnam

Publicity and Social Media Chairs

Tomohiko Taniguchi	Fujitsu Laboratories, Japan
Nguyen Quoc Dinh	Le Quy Don Technical University, Vietnam
Dao Thi Nga	Le Quy Don Technical University, Vietnam

Publications Chairs

Quang Kien Trinh	Le Quy Don Technical University, Vietnam
Mai Ngoc Anh	Le Quy Don Technical University, Vietnam
Tomoyuki Ohkubo	Advanced Institute of Industrial Technology, Japan

Web Chairs

Trong-Thuc Hoang	The University of Electro-Communications, Japan
Vu Hoang Gia	Le Quy Don Technical University, Vietnam

Posters and PhD Track Chairs

Ulrich Kuhne	Télécom Paris, France
Guanghao Sun	The University of Electro-Communications, Japan
Ta Minh Thanh	Le Quy Don Technical University, Vietnam
Le-Nam Tran	University College Dublin, Ireland

Panels Chairs

Mai-Khanh Nguyen Ngoc	The University of Tokyo, Japan
Le Chung Tran	University of Wollongong, Australia
Berk Canberk	Istanbul Technical University, Turkey

Demos Chairs

Zoran Hadzi-Velkov	Ss. Cyril and Methodius University, Macedonia
Van Sang Doan	Kumoh National Institute of Technology, South Korea
Quang Nguyen The	Le Quy Don Technical University, Vietnam

Tutorials Chairs

Jean-Luc Danger	Télécom Paris, France
Duc Anh Le	Center for Open Data in the Humanities, Tokyo, Japan
Tuan Le	Middlesex University, UK

Technical Program Committee

Truong Khoa Phan	University College London, UK
T. Tuan Nguyen	University of Buckingham, UK
Purav Shah	Middlesex University, UK
Tuan Anh Le	Middlesex University, UK
Cong Trang Mai	Queen's University Belfast, UK
Le Chung Tran	University of Wollongong, Australia
Huy T. Nguyen	Nanyang Technological University, Singapore
G. Suseendran	Vels Institute of Science, Technology & Advanced Studies, India
Falowo Olabisi	University of Cape Town, South Africa
Thang Vu	University of Luxembourg, Luxembourg
Yuanfang Chen	Hangzhou Dianzi University, China

Kien Nguyen	Chiba University, Japan
Nguyen Ngoc Mai Khanh	The University of Tokyo, Japan
Guanghao Sun	The University of Electro-Communications, Japan
Duc Anh Le	Center for Open Data in the Humanities, Tokyo, Japan
Tomoyuki Ohkubo	Advanced Institute of Industrial Technology, Japan
Luong Duy Manh	Le Quy Don Technical University, Vietnam
Kien Trinh	Le Quy Don Technical University, Vietnam
Dao Thi Nga	Le Quy Don Technical University, Vietnam
Huu Hung Nguyen	Le Quy Don Technical University, Vietnam
Xuan Tung Truong	Le Quy Don Technical University, Vietnam
Tang Van Ha	Le Quy Don Technical University, Vietnam
Ta Minh Thanh	Le Quy Don Technical University, Vietnam
Doan Van Sang	Kumoh National Institute of Technology, South Korea
Toan Dao	University of Transport and Communications, Vietnam
Huan Vo	Ho Chi Minh City University of Technology and Education, Vietnam
Van-Ca Phan	Ho Chi Minh City University of Technology and Education, Vietnam
Pham Ngoc Son	Ho Chi Minh City University of Technology and Education, Vietnam
Kien Dang	Ho Chi Minh City University of Transport, Vietnam
Toan Doan	Thu Dau Mot University, Vietnam
Dac-Binh Ha	Duy Tan University, Vietnam
Nguyen Gia Nhu	Duy Tan University, Vietnam

Contents

An Attempt to Perform TCP ACK Storm Based DoS Attack on Virtual and Docker Network

Khanh Tran Nam, Thanh Nguyen Kim, and Ta Minh Thanh(✉)

Le Quy Don University, Ha Noi, Viet Nam
noangel0607@gmail.com, thanhcuchp@gmail.com,
thanhtm@mta.edu.vn
http://www.lqdtu.edu.vn/

Abstract. Recently, the server virtualization (hypervisor) market is growing up fast because server virtualization has many benefits. More and more businesses use hypervisors as an alternative solution to a physical server. However, hypervisors are more vulnerable than traditional servers according to recent researches. Therefore, stand on the position of a system administrator, it's necessary to prepare for the worst circumstances, understand clearly, and research for new threats that can break down the virtual system. In this paper, we attempt to perform TCP ACK storm based DoS (Denial of Service) attack on virtual and Docker networks and propose some solutions to prevent them.

Keywords: DoS · Hypervisor · TCP · ACK storm · Virtual network · Docker network

1 Introduction

1.1 Overview

Network security is one important aspect of many aspects that a system administrator is interested in because there are many potential cybersecurity threats to a hypervisor system [9–12]. The DoS/DDoS attack is one of those threats [8]. In 2019, Imperva [1] had reported an SYN DDoS attack in which 500 million packets per second (PPS) in January and another in which 580 million packets per second (PPS) in April. Each of the packets was thought to a median number of 850 bytes per packet. That means that 580 million 850-byte packets would result in about 3944 Gbps of data targeting your network protocol every second to render it unresponsive. Previously, in 2018, the GitHub DDoS Attack was recognized as sustaining a 1.35 Tbps (with 129.6 million PPS) attacks without the help of botnet. "Size" of DDoS attack is increasing year by year and cost businesses thousands to millions of dollars in losses. To prevent and minimize the DoS/DDoS attack's sabotage, analyzing more types of DoS attack

N.-S. Vo and V.-P. Hoang (Eds.): INISCOM 2020, LNICST 334, pp. 243–258, 2020.
https://doi.org/10.1007/978-3-030-63083-6_19

and is necessary. In this paper, we attempt to perform TCP ACK-Storm based DoS attack on virtual and Docker networks. Besides, we propose a new attack method based on ACK-Storm DoS attack with FIN-ACK packets which can make vSwitch/vBridge fall into a state of port-exhaustion in a period of time.

1.2 Our Contributions

In our knowledge, the research on the attacks to the vSwitch/vBridge of hypervisor systems and docker systems is not focused on, especially on the docker. Therefore, real services deployed on hypervisor systems or docker systems are vulnerable to attack via a network. That is the motivation of our paper to research the related network attacks on such systems. In summary, we briefly introduce our contributions in this paper as follows:

1. We propose to attempt DoS attacks using FIN-ACK storm on services deployed on virtual systems so that service providers understand the risks, and security vulnerabilities when deploying the services in a virtual environment. That implies that real applications deployed on virtual systems can be easily attacked by hackers via the Internet.
2. The DoS attacks proposed on the Virtual Machine and Docker systems in this paper is the first attempt that is made to prove feasible when a hacker wants to attack services on a hypervisor system.
3. We propose a new attack method based on ACK-Storm DoS attack with FIN-ACK packet which can make vSwitch/vBridge fall into a state of port-exhaustion in a period of time.
4. Based on these attack experiments, our paper also offers some solutions to prevent and decrease the destruction of these types of attacks in real applications.

1.3 Roadmap

The rest of this paper is organized as follows: In Sect. 2, a brief overview of the related works are presented. We focus on the explanation of the ACK-Storm DoS attack and the FIN-ACK-Storm DoS attack. To illustrate, In Sect. 3 and Sect. 4, the detail of the proposed DoS attacks on the hypervisor systems, VMware and Docker, is explained. Based on our experimental attacks, the simulation results and discussion are shown. Furthermore, In Sect. 5, we assess the feasibility of the Ack-Storm DoS attack on hypervisor systems. Finally, Sect. 6 gives the conclusions of this paper.

2 Related Works

2.1 Transmission Control Protocol - TCP

TCP is a Connection-oriented transmission protocol, it establishes connection channel before transferring data. Through TCP, applications on networked

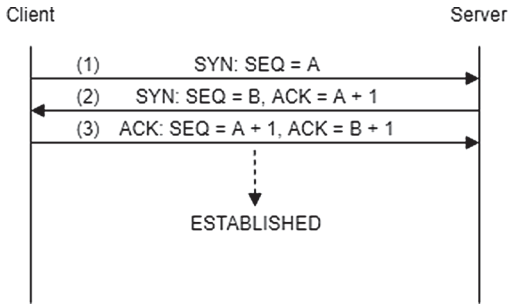

Fig. 1. TCP Three-step Handshake - Establish TCP connection

servers can communicate with each other, through which they can exchange data or packets. This protocol ensures reliable data delivery to the receivers. Moreover, TCP has the function of distinguishing between data of many applications (such as Web services, Email services, and so on) simultaneously running on the same server. The operation of the protocol TCP is described in the RFC-793 [7]. Nowadays, TCP is still widely used in many server systems.

Three-Step Handshake. Three-step handshake, or maybe called a Three-way handshake, is used in TCP to establish a connection. TCP uses passive open, a server bind to and listens to a port before a client tries to connect to the server. A client may start an active open after the passive open is established. The three-step handshake occurring in three steps (see Fig. 1) can be described as follows:

1. The client, who wants to connect to the server, sends a TCP packet to the server with a random value A for the segment's sequence number (SEQ) and a bit SYN set. This packet is called a SYN packet (1).
2. After the server receives the SYN packet, it responds to the client a TCP packet with a random value B for SEQ number, the acknowledgment number is set to one more than the received sequence number (A+1), and two-bit SYN, ACK are set. This packet is called a SYN-ACK packet (2).
3. Finally, the client sends an ACK packet (3), which has the acknowledgment number is set to one more than the received sequence number (B+1), the sequence number is set to the received acknowledgment value (A+1), and only bit ACK set.

Four-Step Handshake. Four-step handshake, or four-way handshake, is used to terminate TCP connection with each side of the connection terminating independently. It occurs as follows:

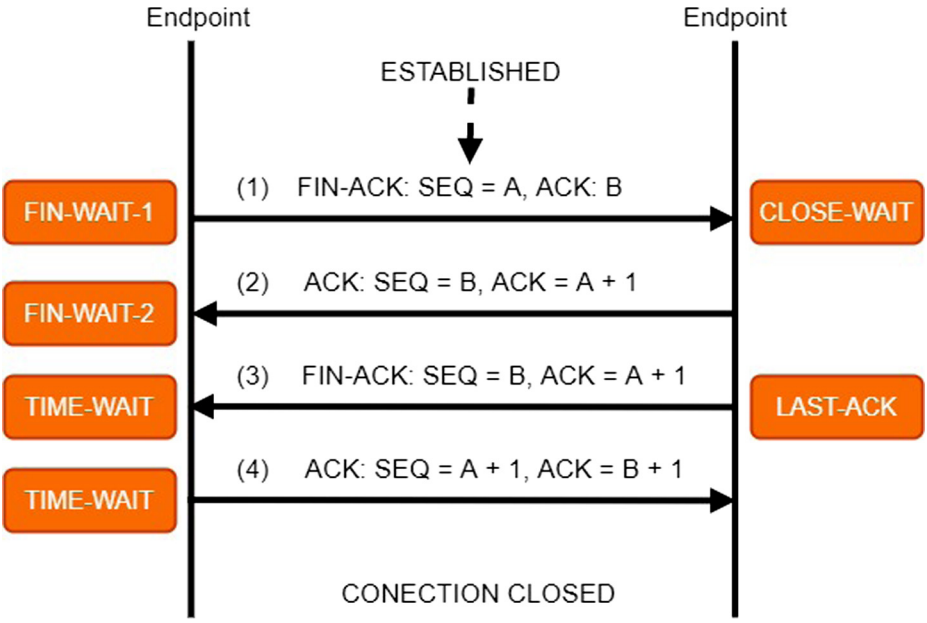

Fig. 2. TCP Four-step Handshake - Terminate TCP connection

1. When one party X of a TCP connection wants to terminate its half of the connection, it sends to the other endpoint a FIN packet, which has FIN bit set, the SEQ number (A), and ACK number (B) depending on the current state of the TCP connection. Then it enters the FIN-WAIT-1 state.
2. The other endpoint Y receives the FIN packet, free up its buffer, and responds an ACK packet, with acknowledgment number is more one than the received sequence number (A+1) and the sequence number is set to the received acknowledgment value (B). It enters the CLOSE-WAIT state. After receiving the ACK packet from endpoint Y, endpoint X enters the FIN-WAIT-2 state from the FIN-WAIT-1 state. The connection from endpoint X to endpoint Y is terminate, but the connection from endpoint Y to endpoint X still opens, this is the half-close connection.
3. Endpoint Y sends a FIN packet to terminate the connection from endpoint Y to endpoint X and wait for an acknowledgment from endpoint X.
4. Finally, when endpoint X receives the FIN packet from endpoint Y, it enters the CLOSED state.

2.2 Vmware Workstation [4]

Virtual Machines (VMs). Virtual machines are software computers that provide the same functionality as physical computers. They are based on computer architectures but they behave as separate computer systems. With them,

the users could run different software requesting different environments without conflict at the same time. They bring many benefits for users and businesses: Reduced hardware costs; Faster desktop and server provisioning and deployment; Small footprint and energy saving; Increasing IT operational efficiency, and so on.

Hypervisor. A hypervisor, called a virtual machine monitor (VMM), can be hardware, software, or firmware that provides virtualization capability. A hypervisor allows one host computer, which the hypervisor operates in, to support multiple guest VMs by virtually sharing its resources such as memory and processing, and so on. According to the resources that have been allocated for each virtual machine, the hypervisor gives and manages the scheduling of VM resources against the physical resources. The hypervisor has two types: type 1,"bare metal", run directly on the host's hardware, like an operation system, while type 2, "hosted", run as software on an operating system, as an application. What type of usage is based on the purpose and need of the user and businesses.

Virtual Switch (vSwitch). A virtual switch is a software application that allows communication between virtual machines, between the physical machine and virtual machines. It directs the communication on a network in an intelligent way by ensuring the integrity of the virtual machine's profile, which includes network and security settings checking data packets before moving them to a destination. A virtual switch is completely virtual and can connect to a network interface card (NIC). The vSwitch merges physical switches into a single logical switch. This helps to increase bandwidth and create an active mesh between a server and switches. It also helps in easy deployment and migration of virtual servers, allows network administrators to manage virtual switch deployed through a hypervisor, and easy to roll out new functionality, which can be hardware or firmware related.

Virtual Network - Network Virtualization. Network Virtualization is a method of splitting up the available bandwidth into channels to combine available resources in a network. Each of the channels can be assigned (or reassigned) to a particular server or device in real-time and is independently secured. The main idea of Network virtualization is that virtualization disguises the true complexity of the network by splitting it into manageable parts, like a partitioned hard drive, making it easier to manage files. Every subscriber has shared access to all the resources on the network from a single computer.

2.3 Docker [5]

Docker Engine - Docker Daemon. Docker Engine, which may be called Docker Daemon, is a background-running service that manages everything

required to run and interact with Docker containers on the host operating system. It's used to run Docker containers which bundled up all application dependencies inside. Docker Engine enables containerized applications to run anywhere consistently on any infrastructure. Docker Daemon communicates directly with the host operating system and knows how to ration out resources for the running Docker containers. It's also an expert at ensuring each container is isolated from both the host OS and other containers. In simple terms, it replaces the hypervisor.

Docker Container Image. Docker container image, or Docker image, is a lightweight, standalone, executable package of software. It includes code, runtime, system tools, system libraries, and setting files - all things needed to run an application. There are many Docker images available that could be used to rebuild new images or deployed Docker containers.

Docker Container. A Docker container is an instance of a deploying Docker image. But we could modify Docker containers. Multiple containers can run on the same machine at the same time and share the OS kernel. They run as independent processes in userspace. Containers take up less space than VMs, can handle more applications, and require fewer VMs and Operating systems.

Docker Network. The Docker networking philosophy is application-driven. Docker network isolation achieved using Network namespace. Typically, Services gets separate IP and maps to multiple containers. Microservices done as Container puts more emphasis on integrated Service discovery. As the Container scale on a single host can run to hundreds, host networking has to be very scalable.

Virtual Bridge (vBridge). A virtual bridge (vBridge), which may be called Network bridge or Linux bridge, is a piece of software used to unite two or more network segments. It works like a virtual network switch (vSwitch) and working transparently [13,14]. In Docker container built with a Linux image base, the Docker network is managed by vBridge.

2.4 Network and Port Address Translation [6]

Network Address Translation - NAT. Network Address Translation (NAT) is a technique that allows one or more internal IP addresses to be converted to one or more external IP addresses. This technique makes a device in a local/private network could connect to the public network (Internet). NAT is responsible for transmitting packets from one network layer to another in the same network. NAT will make changes to the IP address inside the packet. Then move through routers and network devices. On the contrary, when the packet is transmitted from the internet (public) back to the NAT, NAT performs the task of changing

the destination address to the IP address inside the local network and sending it. Moreover, NAT can act as a firewall. It helps users secure computer IP information. Specifically, if the computer is having trouble connecting to the internet, the public IP address (previously configured) is displayed instead of the local network IP address.

Port Address Translation - PAT. Port Address Translation (PAT) is an extension technique of NAT which could help multiple devices on a local/private network connect to the public network by mapping their local IP address to a single public IP address and specific ports. With each set of local-IP:local-port is mapped to a public IP address with a specific port, multiple devices could communicate with the Internet with the corresponding ports provided to them. This technique could conserve IP addresses but the number of ports is not unlimited, only 65,536 ports so there can be a theoretical maximum of 65536 PAT entries at a time for each inside global address. If an attacker can occupy all 65,536 ports, there is a port-exhaustion, and no communications can be made between local devices and the public network.

2.5 Original ACK-Storm DoS Attack

The original idea [2] is suggested by Mr. Abramov and Prof. Herzberg depending on the vulnerable handle exceptions of TCP that described on page 72 of RFC 793 [7] about TCP: when a TCP connection is in the ESTABLISHED state received a packet with not-yet-sent acknowledges data SEG.ACK > SND.NXT (Acknowledgement from the receiving TCP higher than the sequence number of the next byte of data to be sent to the other), the received one handle as follows: Send an ACK (with the last sent SEQ/ACK number) to another party of connection, then drop the segment and return. In particular, ignore the payload in the segment. However, this state has a timeout and stopped when the timer reaches the timeout. For raising the basic ACK-storm DoS attack (Two Packets ACK Storm), the attackers act as the following scenario:

1. Pick up (at least) one packet from a TCP connection between a client and a server (Just need to eavesdrop one packet and do not need any impacts on the connection).
2. Generate two packets, each addressed to one party, and with a sender address of the other party (*i.e.* spoofed). The packets must be inside the TCP windows of both sides. The packets should have content - at least one byte of data or it will not be implemented.
3. Send the packets to the client and the server at the same time. The connection will then enter an infinite loop of sending ACK packets back and forth between both parties.

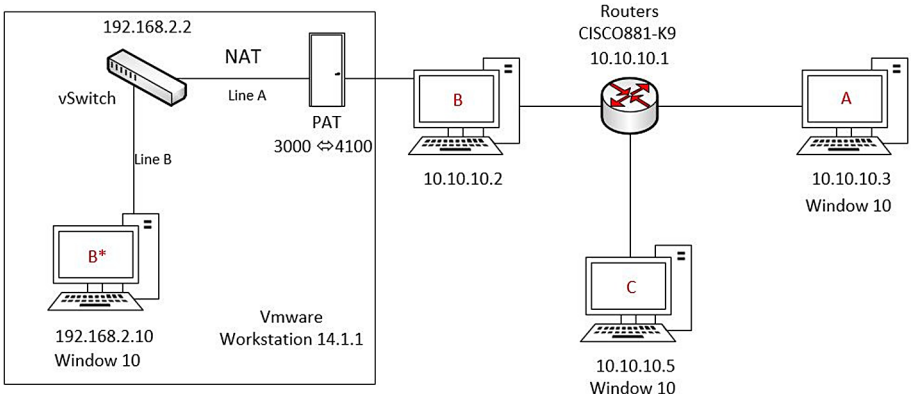

Fig. 3. Experimental model - Three physical computers as above connected with each other by a router Cisco 800 Series Routers CISCO881-K9, the C is attacker which can eavesdrop and inject packets into TCP connection between Host A and Hypervisor B

2.6 ACK-Storm DoS Attack Using FIN-ACK Packet

Depending on Abramov's idea [2], Son proposed another ACK-Storm DoS attack [3] with the same mechanism, but the starting point is sparked by a couple of FIN-ACK packets created by the same way when creating a couple of ACK packets to trigger ACK-Storm DoS attack. According to the description in RFC-793 [7] (p. 73): if a TCP connection is in CLOSE-WAIT state, it does the same processing as for the ESTABLISHED state. That means if attackers force each party running into CLOSE-WAIT, each party waits for an ACK packet (see Fig. 2) never come but not-yet-sent ACK packet instead, then the ACK-Storm DoS raised by two FIN-ACK packets starts. Because no timeout by default for CLOSE-WAIT state, this DoS attack will never stop in theory, and parties of the connection will stay in CLOSE-WAIT state forever.

3 Experiment in VMware Workstation

3.1 Experiment Original ACK-Storm DoS Attack

Environment for Experiment (see Fig. 3**)**

Hypervisor. We use VMware Workstation 14.1.1 installed Windows 10 64 bit for Virtual Machine server B*.

Physical Computers. Host A and Hypervisor B have the same configuration as follows: Window 10 64 bit, Chip Inter® Core™ i7-6700 CPU @ 3.40 GHz, RAM 8 GB, the network interface is a 100 Mbps Ethernet adapter attached to the PCI-E bus. And the attacker C is Windows 10 installed Python.

Router. We use a router instead of the hub in Son's experimental test because, in reality, businesses use the router for establishing their LAN network or connect to the Internet. The router here is a Cisco 800 Series Routers CISCO881-K9.

Attack Execution. We use a simple TCP connection created with socket python (v3.7.3) scripts. For the experiment, we just let A and B* create a connection with a Three-Way Handshake and pick up the last ACK packet from the connection for SEQ and ACK sequence number. Then, in attacker C, we use scapy[1] to create a couple of fake ACK packets with source IP is one party and destination IP is the other. We sent those ACK packets to each of the respective parties.

In Line A. We captured about 55000 retransmitted ACK packets in 60 s while the ACK storm was occurring. This result is similar to the results in previous experiments of Abramov [2] and Son [3] but the number of packets is less because of smaller Ethernet adapter bandwidth. However, task manager still displayed the bandwidth used by VMware NAT services was 0.6 Mbps. This result is much bigger than 120 bytes (two packets) sent by the attacker C.

In Line B. Only the first fake ACK packet, which we created, is forwarded to virtual machine B* through vSwitch. No retransmitted ACK packet is directed to virtual machine B*.

3.2 Experiment ACK-Storm DoS Attack Using FIN-ACK Packet

Environment for Experiment. We use the same environment with the experiment of ACK-Storm DoS attack described above.

Attack Execution. We do the same action with the experiment ACK-Storm DoS attack but we use a couple FIN-ACK packets instead of a couple of ACK packets. It means that we just turn the bit FIN flag to 1 and keep all conditions as the experiment ACK-Storm DoS attack as above.

Analysis. When each party received fake FIN-ACK packets, they respond with invalid retransmitted ACK packets, and the ACK-Storm was begun.

In Line A. We captured about 86000 ACK retransmitted packets in 60 s (about 1434 packets per second) in Line A while the ACK storm was occurring. This result is similar to the results in previous experiments of Abramov [2] and Son [3] but the number of packets is less because of smaller Ethernet adapter bandwidth. However, the task manager still displayed the bandwidth used by VMware NAT services was 0.6 Mbps. This result is much bigger than 120 bytes (two packets) sent by the attacker C. We keep experiment for 24 h and it's still working. This

[1] https://scapy.net/.

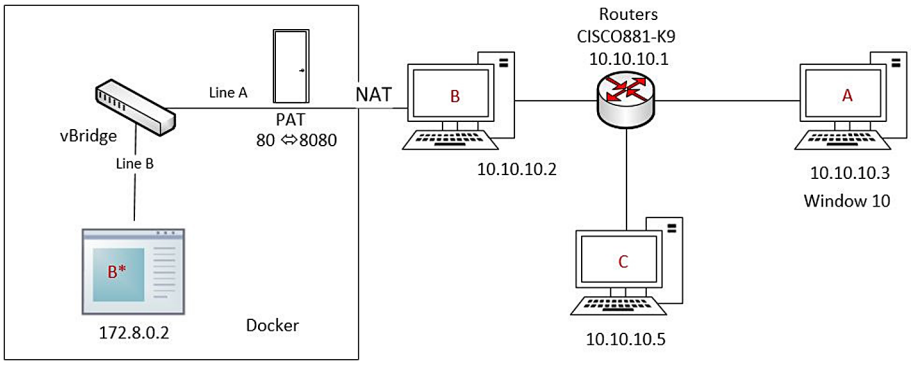

Fig. 4. Experimental model - Three physical computers as above connected by a router Cisco 800 Series Routers CISCO881-K9, computer C is attacker which can eavesdrop and inject packets into TCP connection between Host A and Hypervisor B. Docker container B* use Nginx for server with the local port is 80 and the public port is 8080.

proves that Son's hypothesis [3] is correct. The TCP connection is stuck in the CLOSE-WAIT state while the process established TCP connection keeps running. As the ACK-Storm activated by a couple of FIN-ACK packets can be last forever, the attacker could completely raise DDoS attack to hypervisor B and virtual machine B*. If an attacker could raise a DDoS attack with all available ports, about 65500 ports, he could "play" a DDoS attack which 94 million packets per second (with 40 Gbps), and also cause the port exhaustion.

In Line B. As explained above, in the ACK-Storm DoS attack experiment, only the first fake FIN-ACK packet, which we created, is forwarded to virtual machine B* through vSwicth. Still, no retransmitted ACK packet is directed to virtual machine B*, the same result with the ACK-Storm DoS attack. We assumed that vSwitch responses all retransmitted ACK packets instead of virtual machine B* as long as no RST request is sent between Host A and virtual machine B*. To prove this, we try suspending virtual machine B*, the ACK-Storm attack still going on. We keep experiment for 4 h more and the ACK-Storm attack is no sign of stopping. When we resume virtual machine B*, nothing happens. This is a feature of VMware workstation that prevent all invalid packets to virtual machine inside it.

4 Experiment in Docker

4.1 Experiment Original ACK-Storm DoS Attack

Environment for Experiment

Hypervisor. We use Docker Desktop v2.2.0.5 with Docker Engine v19.03.8, Docker Compose v1.25.4. We build a new image depending on base image Ubuntu 18.04 and install Htop for monitoring the container's activity. We also use TCPdump for monitoring network flow, and use Nginx for server B.

Physical Computers. Host A and Hypervisor B have the same configuration as follows: Window 10 64 bit, Chip Inter® Core™ i7-6700 CPU @ 3.40 GHz, RAM 8 GB, the network interface is a 100 Mbps Ethernet adapter attached to the PCI-E bus. We use Windows 10 for attacker C.

Router We use a router for establishing a LAN network or connect to the Internet. The router here is a Cisco 800 Series Routers CISCO881-K9.

Attack Execution. With TCP connection created by using Chrome to access to the server Nginx in Container B* and do the same as experiment original ACK-Storm DoS attack in VMware Workstation, we pick the last ACK packet in the TCP connection between Host A and Docker container B* for obtaining SEQ and ACK number. Then, we use scapy to create a couple of fake ACK packets with source IP is one party and destination IP is the other. After that, we send those ACK packets to each of the respective parties.

Analysis. When each party received a fake ACK packet, they responded with invalid retransmitted ACK packets, and the ACK-Storm was begun. The ACK-Storm was terminated by Host A (see Fig. 4) after 1 s while TCP connection was timeout after 30 s.

In Line A. We captured about 550 retransmitted ACK packets in 1 s while the ACK storm was occurring. 10 s later, Container B* send an RST-ACK packet. The TCP connection timeout after 30 s. There are still some retransmitted ACK packets generated by the original ACK-Storm attack but the retention time is very short. This is because of Docker's feature (or Nginx's) when the container received many retransmitted ACK packets with the same ACK/SEQ number. This result is not as expected but still proves a flaw in TCP connection, which was discovered by Abramov [2], when receiving not-yet-sent acknowledges data packet.

In Line B. The same result with the experiment ACK-Storm DoS attack, only the first fake ACK packet is forwarded to Docker container B* through vBridge. No retransmitted ACK packet is directed to Docker container B*. It seems like vBridge response all retransmitted ACK packets instead of Docker container B*, same behavior with vSwitch.

4.2 Experiment ACK-Storm DoS Attack Using FIN-ACK Packet

Environment for Experiment. We use the same environment with the experiment ACK-Storm DoS attack in Docker that's described above.

Attack Execution. We do the same with the experiment ACK-Storm DoS attack but we use a couple FIN-ACK packets instead of a couple of ACK packets. It means that we just turn the bit FIN flag to 1 and keep all conditions as the experiment ACK-Storm DoS attack as above.

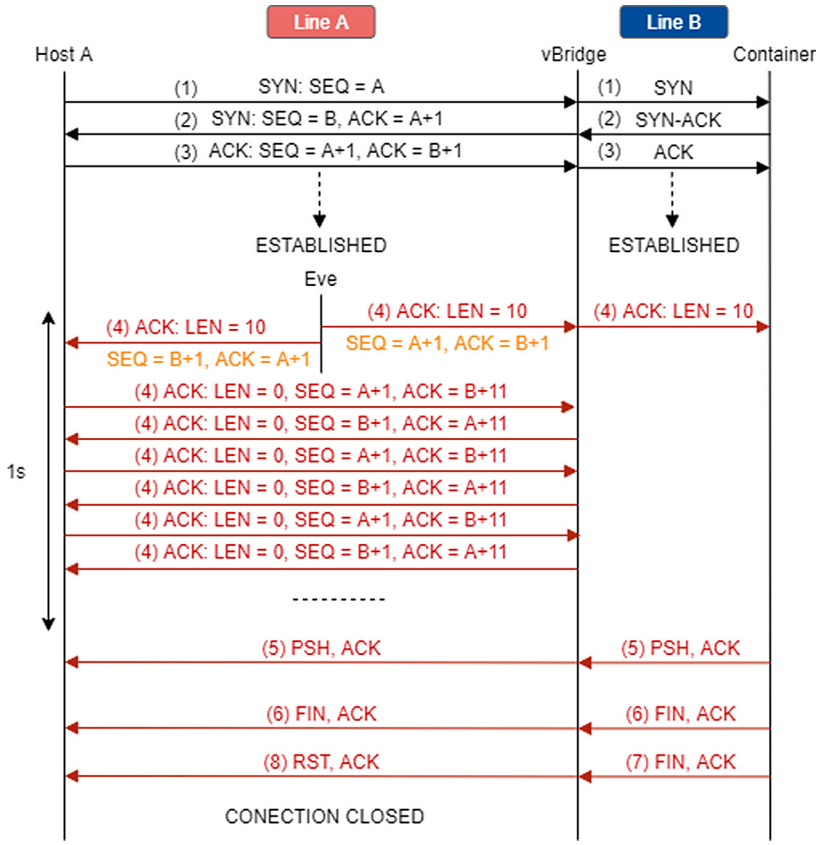

Fig. 5. TCP communication during the experiment in Docker with Ack packet

Analysis. When each party received fake FIN-ACK packets, they responded with invalid retransmitted ACK packets, and the ACK-Storm was begun.

In Line A. We captured a few retransmitted ACK packets between Host A and Container B* before Container B*'s side stopped responding.

In Line B. Same as above, in the experiment ACK-Storm DoS attack, only the first fake ACK packet is forwarded to Container B* through vBridge. No retransmitted ACK packet is directed to Container B*.

Propose. We tried to replace simple ACK packets for attacking by HTTP packets. However, we got the same results as Subsect. 4.1 and 4.2. Because ACK-Storm DoS attack with FIN-ACK [3] packet seem does not work with TCP connection created by browsers like Chrome or Firefox and Nginx, we propose a new attack method based on it: the attacker C create his own fake TCP

connection with server B, send a fake FIN-ACK packet to server B after three-ways handshake completed and constantly send fake retransmitted ACK packets of the same format as retransmitted ACK packets of ACK-Storm DoS attack. Server B, more specifically vBridge, will respond to them and stuck in CLOSE-WAIT state. With a simple TCP connection, the CLOSE-WAIT state has no timeout [7], so we assume server B may be stuck in CLOSE-WAIT state forever. The detail of this attack method will be described in Subsect. 4.3.

4.3 Experiment ACK-Storm DoS Attack Using FIN-ACK Packet and Fake Retransmitted ACK Packets from Attacker

Environment for Experiment. We use the same environment with the experiment ACK-Storm DoS attack in Docker that's described above (see Fig. 5) but no Host A.

Attack Execution. We use scapy (version 2.4.3) to create a fake TCP connection with Container B* by Three Steps Handshake as follows:

1. Use scapy create a fake SYN packet manually with the destination is Container B*, the source is attacker C, ACK number is 0, SEQ number is voluntary, and send to Container B*.
2. Capture response SYN/ACK packet from Container B* then create a fake ACK packet depending on the SYN/ACK packet (use ACK and SEQ number).
3. Send the fake ACK packet to Container B* and the TCP connection is established.

After that, we create a fake FIN-ACK packet as same as the previous experiments and create a fake retransmitted ACK packets like attacker C received FIN-ACK packet similar to Container B*'s. Then, we follow these three steps:

1. Send the fake FIN-ACK packet to the Container B*.
2. Send the fake retransmitted ACK packet to the Container B*.
3. Delay about 1 s and repeat step 2.

Analysis. When Container B* received the fake FIN-ACK packet, it switches to CLOSE-WAIT state and responses with invalid retransmitted ACK packets.

In Line A. Whenever vBridge receives the fake retransmitted ACK packet from the attacker C, it responses with invalid retransmitted ACK packets. While the attacker C keeps sending fake retransmitted ACK packet to the Container B*, the Container B* is stuck in CLOSE-WAIT state. But the attack does not last too long as same as the experiment ACK-Storm DoS attack with FIN-ACK packet [3]. After about 10 min, Container B* sends an RST packet to close the TCP connection. In the experiment, we dump some FIN-PSH-ACK packets but there are no signs of disconnection until the Container B* sends RST packet suddenly. This might be a feature of vBridge when received too many retransmitted ACK packets with the same ACK/SEQ number.

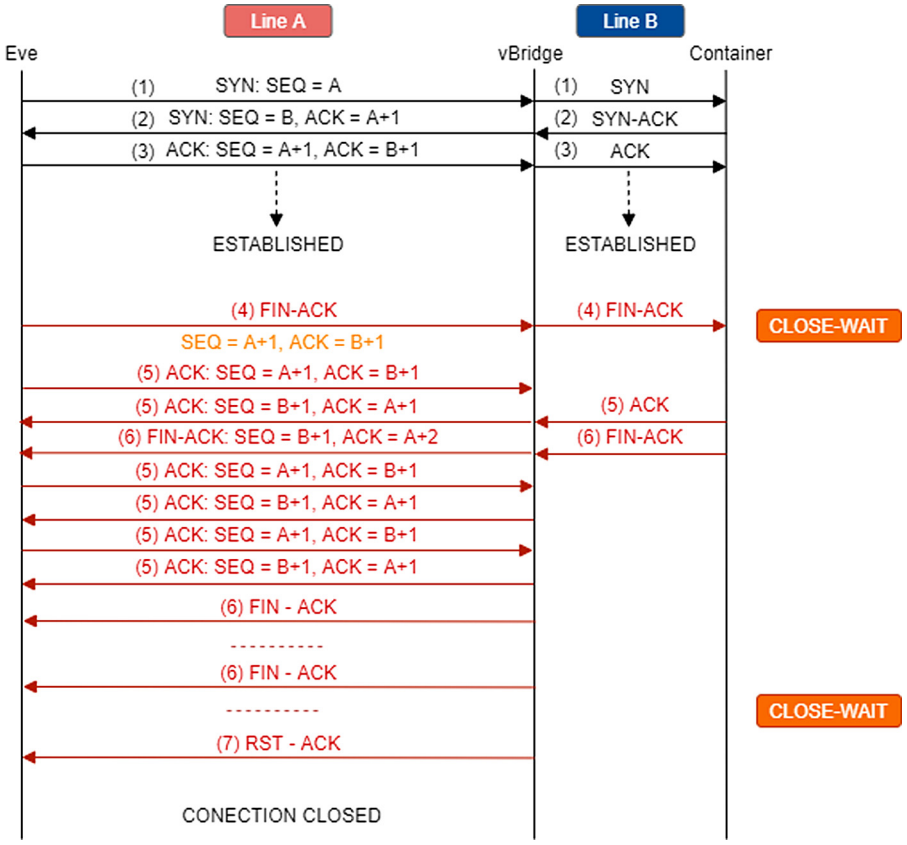

Fig. 6. TCP communication during the experiment using FIN-ACK packet in Docker with fake retransmitted ACK packets from attacker C

In Line B. Only valid packets are directed to Container B*, the same as the result of the experiment ACK-Storm DoS attack. No retransmitted ACK packet is directed to Container B*. It means that vBridge response all retransmitted ACK packets instead of Container B* as long as no RST request is sent between Host A and Container B* (Fig. 6).

5 Discussion

5.1 Feasibility

From the results of the above experiments, we have concluded that the ACK-Storm DoS attack is possible with basic TCP connections in a virtualized environment. Based on the vulnerability discovered by Abramov [2] described in RFC 793 [7] and based on the idea of developing a method of attack with FIN-ACK packet of Son [3], an attacker can perform an attack which is made by creating

a fake connection and sending packets just like a normal TCP connection being attacked by ACK-Storm DoS using FIN-ACK packet.

In our experiment, this attack method is feasible even with virtualization environments like Docker with Nginx for the server. However, in the default configuration of Nginx, each state of the connection has a timeout setting so the attack method using ACK packets is blocked and the risk of being attacked by an attacker with the FIN-ACK packet is also limited. However, attacker C can use a simple python script with scapy to create many connections to the server on the Docker container that uses Nginx to occupy ports within 10 min and can cause an issue, which is called port exhaustion, with vBridge when they combine with a botnet. The time, just about 10 min, is not too long, but the damage to businesses will not be small even though the cost of conducting the attack is not very high. With this attack method, attackers do not need to eavesdrop TCP connection but still can occupy port.

5.2 Countermeasures

As we mentioned above, in Sect. 3, the vSwitch (or the vBridge) responds (or ignores) invalid packets instead of the hypervisor (or the container) and only directs valid packets to the hypervisor. Therefore, the hypervisor will receive no packets from the client during the ACK-Storm DoS attack. So, we can create a timer running on another thread to count the time unresponsive from the client and close that connection when "timeout". Besides, we can create a duplicate-ACK-checker to count the number of duplicate retransmitted ACK packets, when the number reaches the maximum, the hypervisor creates its own RST-ACK packet to force the connection stop completely.

5.3 Ethical Considerations

We disconnected the Internet when conducting our experiments and only tested attacks with the lab computers during the process. These computers only connected to each other during the experiment and did not connect even to the university local area network. Therefore, our experiments are completely harmless to the Internet.

6 Conclusions

For well-known software built by professional teams, such as Nginx, this type of attack is difficult to perform. However, its implications for the virtual server system are still evident once it is successfully implemented. This paper emphasizes the proper attention to handling timeout vulnerabilities with CLOSE-WAIT state in TCP connection when building and developing new software using TCP connection. This flaw could lead to an unlimited ACK-Storm DoS which could harm servers. Finally, TCP is still an important protocol and is used in many software, it was built long ago so inevitably there are vulnerabilities, so find the

flaws and carefully study the vulnerabilities to find out methods to prevent and fix these gaps are the necessary work of all information security experts. In this article, we have focused on researching and analyzing deeply the ACK-Storm DoS attack models of Abramov [2] and Son [3] as well as proposing a new attack version to capture the network ports of a virtual server. In the future, we will focus on researching other virtualized server systems such as Hyper-V, Oracle, an so on, with these types of attacks as well as proposing new attack measures and effective countermeasures.

References

1. Imperva's DDoS Attack reports. https://www.imperva.com/blog/this-ddos-attack-unleashed-the-most-packets-per-second-ever-heres-why-thats-important. https://www.imperva.com/blog/2019-global-ddos-threat-landscape-report
2. Abramov, R., Herzberg, A.: TCP ACK storm DoS attacks. In: Proceedings of the 26th IFIP TC 11 International Information Security Conference, SEC 2011, pp. 12–27, June 2011. https://www.researchgate.net/publication/225532285_TCP_ack_storm_DoS_attacks
3. Duc, S.N., Mimura, M., Tanaka, H.: An analysis of TCP ACK Storm DoS attack on virtual network. In: 2019 19th International Symposium on Communications and Information Technologies (ISCIT), Ho Chi Minh City, Vietnam, pp. 288–293 (2019). https://ieeexplore.ieee.org/document/8905220
4. VMWare: Workstation for Windows, VMWare Workstation Pro 14. https://www.vmware.com/products/workstation. Accessed 10 Apr 2020
5. Docker: Docker desktop for Windows, Docker Nginx. https://www.docker.com/. Accessed 20 Apr 2020
6. Bansal, A., Goel, P.: Simulation and analysis of network address translation (NAT) & port address translation (PAT) techniques. Int. J. Eng. Res. Appl. **7**(7, Part 2), 50–56 (2017). http://www.ijera.com/papers/Vol7_issue7/Part-2/I0707025056.pdf. ISSN 2248–9622
7. RFC 793 - Transmission Control Protocol, DARPA Internet Program, Protocol Specification, pp. 72–73, September 1981. https://tools.ietf.org/html/rfc793
8. Chelladhurai, J., Chelliah, P.R., Kumar, S.A.: Securing docker containers from Denial of Service (DoS) attacks. In: 2016 IEEE International Conference on Services Computing (SCC), San Francisco, CA, pp. 856–859 (2016)
9. Blenk, A., Basta, A., Reisslein, M., Kellerer, W.: Survey on network virtualization hypervisors for software defined networking. IEEE Commun. Surv. Tutor. **18**(1), 655–685 (2016)
10. Bauman, E., Ayoade, G., Lin, Z.: A survey on hypervisor-based monitoring. ACM Comput. Surv. **48**, 1–33 (2015)
11. Chowdhury, N.M.K., Boutaba, R.: A survey of network virtualization. Comput. Netw. **54**(5), 862–876 (2010)
12. Fischer, A., Botero, J.F., Beck, M.T., de Meer, H., Hesselbach, X.: Virtual network embedding: a survey. IEEE Commun. Surv. Tutor. **15**, 1888–1906 (2013)
13. Arch Linux: Network Bridge. https://wiki.archlinux.org/index.php/Network_bridge
14. Varis, N.: Anatomy of a Linux bridge. In: Proceedings of Seminar on Network Protocols in Operating Systems, p. 58 (2012). https://wiki.aalto.fi/download/attachments/70789083/linux_bridging_final.pdf