Electronic Preprint for Journal of Information Processing Vol.30

Regular Paper

Leverage Slow-port-exhaustion Attacks by Exploiting Abnormal Connections from IoT Devices and Docker Containers

Son Duc Nguyen[1,a)] Mamoru Mimura[1,b)] Hidema Tanaka[1,c)]

Received: October 20, 2021, Accepted: May 9, 2022

Abstract: Recently, with the proliferation of IoT devices, network technologies have also rapidly developed to serve the rising needs of users. IoT devices are often complemented by cloud computing technology to provide better services. Fog computing was introduced as a method to bring cloud applications closer to IoT devices so that end-users could avoid communication latency. An edge device at the fog node could use Network Functions Virtualization to optimize its performance and resource management. However, recent research has shown that certain fundamental virtual switch settings can be misused to carry out cyberattacks. In previous research, we proposed Slow-port-exhaustion DoS Attack, an attack that targets virtual switches using the Port Address Translation mechanism for communication between virtual machines and the physical network. In this attack, an attacker with a low amount of attack bandwidth can sabotage the virtual switch by occupying all of the host machine's ports for a long period of time. In this paper, we introduce some methods for exploiting IoT devices to leverage this attack. We also perform experimental attacks with new methods and compare the results with the old methods. Finally, we suggest some countermeasures against this kind of attack.

Keywords: virtual network, security testing, DoS attack

1. Introduction

The emergence of the Internet of Things (IoT) has been rapidly changing the network industry. The concept of the IoT network, which allows all things to connect to the Internet, is an intelligence network that can exchange information and communicate through information sensing devices by using agreed upon protocols [1]. To optimize production efficiency and help minimize environmental impact and downtime, the data from the IoT devices are often transferred to cloud server for process [2]. However, with a myriad of IoT devices such as mobile phones, cars, sensors connecting to the cloud, this transmission faces many challenges like unpredictable latency and privacy gaps. As an alternative solution, fog computing was proposed to address some of the limitations of cloud computing.

Fog computing is defined as a highly virtual platform that provides computing, storage, and networking services at the edge of the network [3]. Fog computing model has been recognized as edge computing including Multi-Access Edge Computing [4], cloudlets [5], and Vehicular Ad Hoc NETworks (VANET) [6]. The principal idea in all these architecture is to make it possible to run applications on virtualized hardware devices that are near to the end-users. The key component of fog computing is the fog nodes, which are edge devices that can provide processing and storage capabilities at the edge of the network [7]. A basic edge device contains the physical hardware of CPU, memory units, and network interface, which operates along with its corresponding OS system. However, edge devices often utilize virtualization to extend their capability in resource management for better performance.

Marin-Tordera et al. summarize edge device virtualization system into 3 types [8]. The virtualization can be handled by a native hypervisor such as VMWare ESXi [9], or by a hosted hypervisor like VirtualBox [10]. Another approach is virtualization on the operating system level by using a known platform called Docker to provide virtual images and containers [11]. Each virtualization type has its own advantages and drawbacks. Apparently, a downside of virtualization is that if the hypervisor or the virtualization software is broken down by an adversary, all of the applications and virtual machines running on that host will become unavailable to users. Recent research shows that virtualized systems are vulnerable against side-channel attacks or Denial-of-Service (DoS) attacks [12], [13], [14].

DoS attack is a well-known type of cyberattack that makes the targeted machine or network unavailable to its users. Common DoS attack methods are flooding the target's network layer by sending overwhelming amounts of network traffic [15]. Another DoS approach is using a small stream of very slow traffic to occupy the target's resources. These methods are called Slow DoS attacks and they are difficult to detect because they generate traffic that is seemingly legitimate ordinary traffic [16]. To mitigate these attacks, it is essential to understand and discover vulnerabil-

[1] National Defense Academy of Japan, Yokosuka, Kanagawa 239–0811, Japan
[a)] ed19006@nda.ac.jp
[b)] mim@nda.ac.jp
[c)] hidema@nda.ac.jp

ities that can lead to a DoS attack. However, there appears to be a lack of academic research regarding Slow DoS attacks, especially Slow Dos attacks that target virtualized systems.

In our initial work, we proposed a Slow-port-exhaustion (SPE) attack, a Slow DoS attack type that targets hosted hypervisor implementing Port Address Translation (PAT) for its network connection [17]. The SPE attack can occupy all of the hypervisor's available ports and thereby prevent users from connecting to their VM. We also proposed 2 different methods, which are called *keepalive-type* and *closewait-type*, to maintain the connections for a long time. Our experimental attack results on VMware Workstation Pro v14.1.2 hypervisor [19] showed that these attacks can sabotage the virtual switch for a long time and prevent legitimate virtual machines from establishing a new connection with the physical network.

Our recent work however, reveals that the proposed methods have certain drawbacks and might not work on recent hypervisor versions [18]. Particularly, some recent hypervisors have timeout settings to terminate half-open TCP sessions created by the prior attack methods. Therefore, in this paper, we introduce novel methods to address the problems. By misusing some IoT devices with abnormal network settings discovered in [18], we propose a new attack scenario by which the attacker can easily take over a large number of TCP connections for a long period of time. We also propose another novel scenario of using Docker containers to assist in SPE attacks. Our testing results on VMware Workstation Pro version 16.1.2 show that our new methods are superior to the older SPE methods.

In summary, we make the following contributions:

(1) We investigate IoT devices that allow TCP connection hanging for a long time and propose a method for using them for the SPE attack.
(2) We show abnormal behavior in the network transmission of the Docker containers and use it for a novel SPE attack scenario.
(3) We show experimental attack results on a recent hypervisor and analyze the results.

This paper is organized as follows: Section 2 reviews the research and technology relating to this research. Section 3 describes all of our SPE attack methods from the older methods to the novel idea. Section 4 shows our testing experiments on the recent version of the VMware Workstation hypervisor and discusses the results. We suggest some countermeasures in Section 5. Finally, Section 6 summarizes our findings and discusses future work.

2. Related Work and Technology

2.1 Virtual Switch with Port Address Translation

A hypervisor is the software or firmware that creates and manages virtual machines. Native (or bare-metal) hypervisors operate directly on the hardware of the host machine while hosted hypervisors are installed as a virtualization software on the host machine OS [20]. The hosted hypervisor provides its virtual machines an operating platform and supervises the execution of the virtual machines. While native hypervisors are widely used in large companies and cloud computing, small instituions and

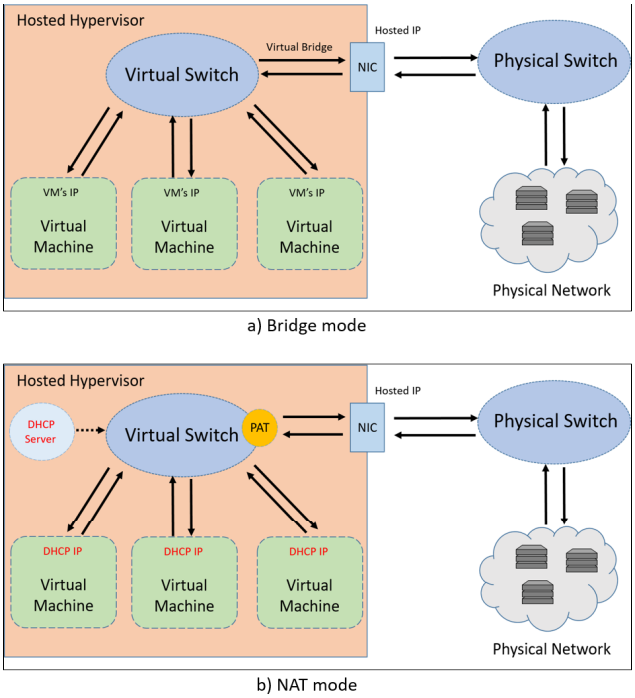

Fig. 1 Virtual switch configurations.

workgroups often use hosted hypervisors because of their simplicity in building and managing virtual machines without requiring another management machine. Some notable examples of hosted hypervisors are Oracle VirtualBox [10], VMware Workstation [19], Parallels Desktop for Mac [21], and QEMU [22].

In this research, we focus on the Network Functions Virtualization (NFV) of the hypervisors. Generally, the hypervisor utilizes a *virtual switch* to manage its network connection. The virtual switch is a software application that is embedded within the hypervisor and performs checking and transmitting network packets. The virtual switch can communicate with other devices on the physical network through a single or multiple physical network interface cards (NICs). The virtual switch also assigns *virtual network adapters* to the virtual machines. The virtual network adapters are responsible for sending and receiving network packets from their corresponding virtual machines.

The virtual switch often has 2 configurations for network connections between the virtual machines and the physical network. One configuration is called *Bridge mode* while the other is called *NAT mode*. **Figure 1** illustrates these configurations. In the Bridge mode, the virtual switch utilizes a virtual bridge function to directly connect the virtual machines to the physical network. In this mode, the virtual machines in the virtual network can get their IP addresses from the DHCP server in the physical network.

The NAT mode on the other hand uses a virtual DHCP server embedding with the hypervisor to assign the virtual machine's IP addresses. For communication with the external physical network, the virtual switch utilizes Port Address Translation (PAT), which is an extension of the Network Address Translation (NAT) technology [23]. In this configuration, when a virtual machine sends a network packet to a physical machine in the external network, the virtual switch translates the virtual machine's IP address to the hosted machine's IP and allocates a port number used

Table 1 Example of PAT translation table.

Protocol	Inside Local IP : Port	Inside Global IP : Port	Outside Local IP : Port	Outside Global IP : Port
TCP	192.168.76.129:10000	10.35.3.95:30124	10.35.2.234:80	10.35.2.234:80
TCP	192.168.76.129:11000	10.35.3.95:31436	10.35.2.234:80	10.35.2.234:80
TCP	192.168.76.129:12000	10.35.3.95:46527	10.35.2.234:80	10.35.2.234:80
TCP	192.168.76.129:10000	10.35.3.95:24634	10.35.2.234:515	10.35.2.234:515
TCP	192.168.76.129:10000	10.35.3.95:52261	10.35.1.192:80	10.35.1.192:80

for this connection. The host machine then creates a new packet with the source port as the allocated port number and sends it to the destination machine. Therefore, the virtual switch in NAT mode can be considered as an intermediary server between the virtual machines and the physical network [17].

Table 1 shows an example of the PAT translation table when a virtual machine attempts to connect with different physical TCP servers. In this table, the inside local IP is the virtual machine's IP address, while the inside global IP is the hosted machine's IP address. On the other hand, the outside local IP and the outside global IP are the IP addresses of the destination machines in the physical network. When a virtual machine establishes a connection with a physical machine, PAT checks the Inside Local IP address and source port combination, and then assigns a corresponding private port number used for packet transmissions from and to the Outside Global IP. The assigned port number is randomly selected from an ephemeral port numbers pool that the hypervisor is authorized to use. Those ephemeral port numbers must not be configured for any particular protocol by the hosted machine. PAT allows the virtual switch to easily identify the corresponding virtual machine for returning packets from the physical servers. In this way, many virtual machines can share one hosted machine's IP address for connections with the physical network [17].

However, a noticeable drawback to PAT is the limited number of ports. Since the port number is 16 bits, theoretically there are 65,535 ports available for network communication [24]. In a NAT mode virtual switch, if all ports in the ephemeral port numbers pool are being used, then no more ports are available for translation of a new connection. We called this event *port exhaustion* [17]. However, this is very rare because each connection usually has a timeout and automatically closes after the timeout is reached. Window's default TCP connection timeout is 72 seconds but the timeout could be configured differently in each network system [25].

2.2 DoS Attack on Virtual Systems

In 2012, Shea and Liu demonstrated that DoS attacks can do more damage to a virtual network than a physical network [14]. Their test results show that a simple TCP SYN Flood attack can cause a 50% of the performance degradation of a virtual server when compared to a physical server using the same resources. Somani et al. discovered another effect of DoS attacks on cloud computing. When the resources of a virtual machine were overwhelmed by a DoS attack, they observed that all other VMs and servers in the same cloud were also affected by the DoS attack. They also demonstrated that some basic features of cloud computing such as multi-tenancy, auto-scaling, and isolation multiplied the impact of a DoS attack in the targeted cloud [12]. Ristenpart et al. exposed a new threat scenario when attackers try to place their malicious virtual machine on the same host machine as their target in the cloud. Their testing on Amazon's EC2 shows that once attackers achieve that co-residence state, they can perform a number of cross-VM attacks such as side-channel or DoS attacks [13].

In 2017, we discovered that attackers could abuse some anomalies in the default network implementation of the virtual switch for DoS attacks [26]. In particular, the tested virtual switch had a relatively high TCP retransmission frequency. Furthermore, in some conditions, the virtual switch kept retransmitting TCP packets without a timeout interval. To utilize those two irregular retransmissions, we proposed two DoS attack schemes. One is an amplification DoS attack that uses a malicious virtual machine inside the virtual network and the other is a collaboration attack using both insider virtual machine and Man-in-the-Middle method [27]. These works showed that some improper settings of the virtual switch could become possible vulnerabilities for DoS attacks.

We continued this line of work by introducing a novel Slow DoS attack method called Slow-Port-Exhaustion (SPE) attack that targets NAT mode virtual switch [17]. This attack is unlike the previous amplification attacks because it uses a very small stream of data instead of flooding the victim with a traffic occupying a large bandwidth. As a result, the SPE attack is harder to detect than the previous amplification attacks.

2.3 Sockstress Attack

Our SPE attack methods are inspired by a slow DoS method known as the Sockstress attack. This attack is developed by Louis from abnormalities discovered while performing network testings in 2008 [28]. The Sockstress attack abuses TCP Servers with long TCP timeouts to generate indefinite TCP connections.

Figure 2 describes the Sockstress attack sequence. This attack contains 4 steps:

- Step 1: the attacker establishes a normal TCP connection with the target server.
- Step 2: the attacker sets the TCP Window size of the last ACK packet to 0 and sends it to the server. The TCP window size signifies how much data the client is willing to receive at that moment. A zero TCP window size packet suggests that the client's buffer is full and that the server must stop sending more data until further notice.
- Step 3: the server initiates queries to check if the client can accept new data. The attacker keeps replying to the queries with the zero TCP window size packets, which suggest that the client is not ready to take any new data. This creates an

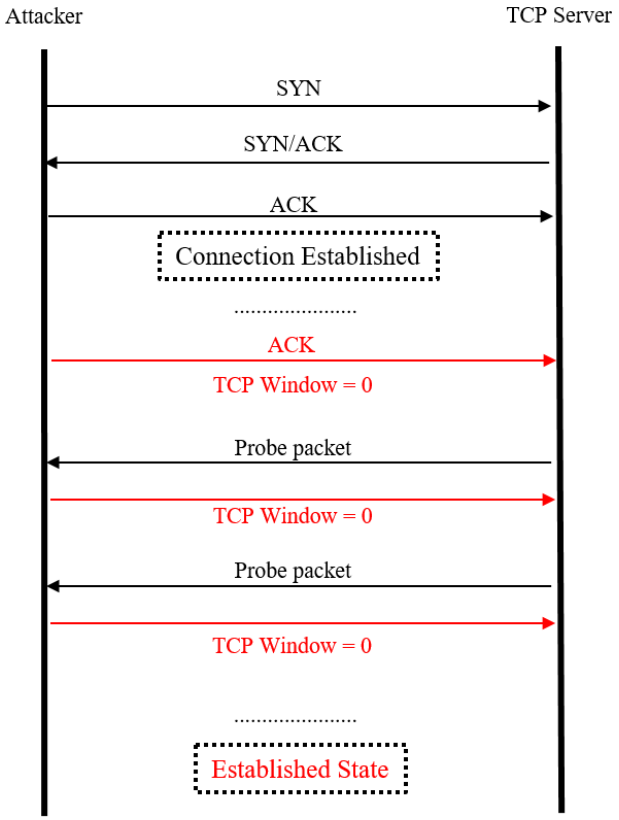

Fig. 2 Socktress attack sequence.

indefinite TCP connection that does not close after a timeout interval.

- Step 4: the attacker generates a large number of indefinite TCP connections to flood the target's resources and prevent legitimate users from establishing a new connection with the targeted server.

This attack relies on a flaw in the implementation of the TCP specification to create indefinite TCP connections. This concept inspires us to find other methods to generate such TCP connections. In 2017, we developed the SPE attack, which used 2 different methods to create indefinite TCP connections and consume the targeted virtual switch's resources [17].

3. SPE Attack Methods

3.1 Attack Model

The SPE attack aims to use a very limited amount of attack bandwidth to occupy all available ports that the hosted hypervisor is able to use for its network connection. Since it is very different from the common DoS attacks approach which is rapidly flooding the victim's network with high bandwidth traffic, the SPE attack can be considered as a Slow DoS Attack [17]. However, the attacker needs a compromised virtual machine in the target's virtual network. In this paper, we consider the attacker is a user of one virtual machine in the targeted virtual network. An example of our attack scenario is organizations using Virtual Desktop Infrastructure (VDI) to manage the virtual desktops and utilize the thin clients in their network. Since our methods specifically work on hosted hypervisors using NAT-mode virtual switch, the attack scenario is restricted to organizations using such hypervisors in their servers. In this scenario, we consider the attacker as an insider of the targeted organization. In the real-world scenario, this refers to the threat of an insider attack. A survey conducted in 2018 shows that 53% of surveyed organizations had confirmed insider attacks against their systems [30].

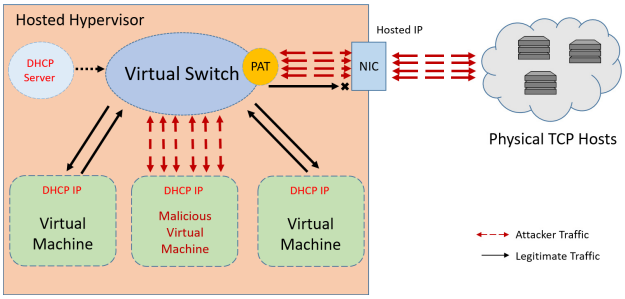

Fig. 3 SPE Attack model.

We present our attacker model in **Fig. 3**. In our attack scenario, the attacker possesses an available virtual machine in the targeted system. In the VDI scenario, the insiders can access their own virtual machines that were registered with their ID. If the attacker wants to create a new virtual machine for attack purposes, they need to have access to the hypervisor in the targeted server. The attacker might not care about whether the attacking virtual machine is identified after the attack if the attacker could manage to use the IDs of the attacker's colleagues to login into their virtual machines and initiate the attack. The attacker's purpose is to prevent all other virtual machines from connecting to the external network. The main approach of the attacker is to force the hypervisor to use up all of its available ports by keeping a large number of TCP connections running indefinitely. The attack sequence is divided into 2 stages. Stage 1 is generating a large number of established TCP connections. Stage 2 is maintaining the TCP connections and not allow the virtual switch to terminate the connections [17].

3.2 Establishing TCP Connections

The attacker establishes the TCP connections with the normal TCP three-way handshake. First, from the compromised virtual machine, the attacker sends `SYN` packets to the listening TCP hosts. The TCP Hosts then respond with `SYN/ACK` packets. After received the `SYN/ACK`, the attacker sends `ACK` packets to the TCP hosts. This sequence will establish the TCP connections [17]. As mentioned in Section 2.1, by changing the source port number or the destination IP address in the `SYN` packet, the attacker can avoid duplication of the port being assigned for translation, thus forcing the virtual switch to use different ephemeral port numbers.

DoS detection methods are normally based on the difference between attack traffic and legitimate traffic [29]. However, it is difficult to distinguish the attack traffic from the normal traffic if the bandwidth of attack traffic is significantly small. Because the SPE attack's goal is to stealthily establish a large number of TCP connections, the amount of `SYN` packets sent to the TCP hosts must be considered small in terms of the "small amount of bytes sent per second". Therefore, our strategy is dividing the `SYN` packets into *waves*. In each wave, the attacker establishes a constant number of connections by sending a fixed number of

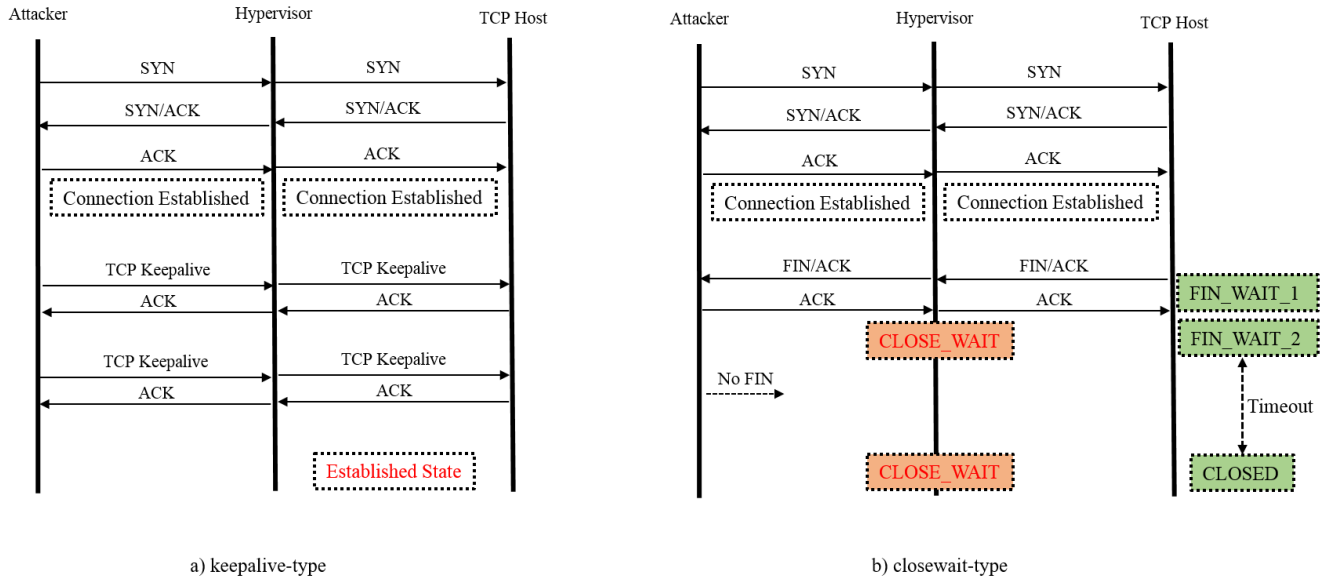

Fig. 4 The old methods of maintaining the TCP connections.

SYN packets using different combinations of source port numbers and destination IP addresses simultaneously.

3.3 Maintaining the Connections

3.3.1 The Old Methods

In this attack model, the attacker could not use the Sockstress method to keep the TCP connections alive because the hosted hypervisor replaces the TCP window size value in the packet to match its own data processing capacity. In our previous research, we proposed 2 different methods to replace the Sockstress method, which we named *keepalive-type* and *closewait-type* attacks [17].

Figure 4 demonstrates the older methods. In the keepalive-type method, the attacker uses TCP keepalive packets to maintain the established TCP connections. The TCP keepalive is a special probe packet that is used to check whether the TCP connection is still working or if it has dropped. A TCP server that enables the keepalive feature usually sets a timer after a connection is established. This timer also resets after the server received a new packet from the other end. After an amount of time depending on the settings of the server, it sends an ACK packet with no data in it to the other end. If the connection is still alive, the other end will also respond with a no data ACK packet. If the other end does not return any packets or sends a RST packet, the server decides that the connection is dropped and terminates this section [31].

In this SPE attack method, after established the TCP connections, the attacker continuously sends keepalive packets to the external physical TCP hosts connecting to the attacker's VM [17]. Because a TCP host could have the option to allow the keepalive feature or not, the attacker must select the physical TCP hosts having their keepalive features enabled.

The closewait-type method to keep the TCP connection alive is putting it in an indefinite CLOSE_WAIT state. This method is based on an unspecified rule about closing the connection in the TCP specification. When the client sends a FIN packet to perform an active close, the server replies with an ACK packet. The server's TCP state will change from ESTABLISHED state to CLOSE_WAIT state. After waiting for the local application to terminate the corresponding process, the server then sends its own FIN packet and moves to LAST_ACK state. However, if the local application does not send the FIN packet, the server will stay on this CLOSE_WAIT state until the connection is forced to close [32]. However, since RFC 793 does not specify the CLOSE_WAIT timeout, the server can remain in this state indefinitely [17].

To exploit this flaw, after receiving the FIN/ACK packets from the TCP hosts, the compromised virtual machine responds with an ACK packet but does not send back its own FIN packet. If the hosted machine does not automatically terminate the process, the sockets will stay in the CLOSE_WAIT state for a long time [17]. Although the sockets are not in the ESTABLISHED state, the ports are still being occupied and the virtual switch can not use those ports to establish new connections.

3.3.2 The New Methods

Both of the older methods have their own drawbacks. The keepalive-type method has to continuously send TCP keepalive packets to maintain the connections. The keepalive-type is feasible with a hosted hypervisor having an average low number of ports the virtual switch can use. In our previous research, we observed that the maximum number of ports that the VMware Workstation version 14.1.2 hosted hypervisor can use is 3,300. As a consequence, our keepalive-type SPE attack only needed to send a low keepalive bandwidth, which is smaller than 1 KBps, and therefore was able to maintain its stealth ability [17]. However, in the scenario of the hosted hypervisor that is allowed to use over 10,000 ports, the keepalive traffic can be detected as attack traffic and the attack is therefore exposed.

The closewait-type method's drawback is that it has to rely on the hosted hypervisor implementing a default TCP configuration without the CLOSE_WAIT timeout. Although RFC 793 does not specifically specify about the CLOSE_WAIT timeout, servers should have stricter TCP settings so that indefinite TCP connection should not occur. In that scenario, the closewait-type SPE

Table 2 Notable TCP hosts and services keeping the ESTABLISHED connection after 24 hour.

Device	Port service (number)
printer	http(80), printer(515), ipp(631)
server	sunrpc(111), printer(515)
server	finger(79), http(8081)
PC	msrpc(135), vmrdp(2179), rdp(3389)
server	echo(7), http(80), sunrpc(111), https(443)
NAS	applications(8200, 10000)

attack will fail since the hosted hypervisor will automatically terminate the hanging `CLOSE_WAIT` connections after a timeout interval. In our preceeding work, we confirmed that this method fails on the recent hypervisor [18].

To address these drawbacks, we proposed two different methods to maintain the TCP connections. The first method is exploiting IoT devices that have simple network settings. While performing scanning in our local network, we discovered some TCP services that maintain TCP connections in `ESTABLISHED` state without sending or receiving TCP keepalive packet. These unusual TCP connections come from printers, network-attached storage, mini server, and personal computers. After 1 hour, we observed that these TCP hosts still let the TCP connections hang in the `ESTABLISHED` state. This type of TCP host behavior obviously assists the SPE attack because the attacker does not have to send the keepalive packets to maintain the connection. The attacker only needs to establish connections with the TCP hosts possessing this unusual setting and let the TCP connections hang in the `ESTABLISHED` state. We called this method *simpleIoT-type*.

We defined an IoT device as abnormal when it has a TCP service maintaining the connection in `ESTABLISHED` state after 24 hours. To identify all of these abnormal IoT devices in our local subnet, we first used Zenmap, which is the well known nmap tool with GUI [39], to conduct SYN scans for available TCP hosts in our subnet. We performed the scans at the same time of day for 5 days. Then we picked up and identified the IoT devices with TCP services that are always available at that time. From the available devices pool, we performed another scan with a self-made Python tool to identify the targeted TCP hosts that have a long established TCP connection. The scan method is simply establishing a TCP connection with each of the TCP services in our subnet and does not initiate active close. We used TCPView, which is a program that shows detailed listings of all TCP and UDP endpoints on the host machine [33], to monitor all of the TCP states generated by our scanner. After 24 hours, we recorded all of the TCP hosts that still keeping the TCP connection in `ESTABLISHED` state.

Surprisingly, with over 99 physical devices available in our local network, there are 58 devices possessing the abnormal TCP connection, which is 58%. On the other hand, there are 117 TCP services having this setting over a total of 484 TCP services available in our subnet. **Table 2** shows some notable TCP hosts and services from our scan result that keep the `ESTABLISHED` state connection after 24 hours. For instance, some specific printers connecting to our subnet provide HTTP and internet printer service. In addition, multiple servers, network-attached storage, and personal computers opening their TCP ports allow the indefinite `ESTABLISHED` connection.

The second novel method to maintain the connection is exploiting TCP servers from Docker containers. We called this method *container-type*. **Figure 5** demonstrates the packets transfer between the attacker's virtual machine, the hypervisor, and the TCP server from a Docker container while using this method. Docker is another virtualization platform that provides the environment to run applications in packages called containers [11]. The containers are more lightweight and more efficient than VMs since they only virtualize the operating system instead of the entire physical hardware [34]. In this SPE method, we created a default TCP server by using the *nginx* image from the Docker hub [35]. The attack sequence is the same with the *closewait-type* method. However, after the TCP server received an `ACK` from the hypervisor and went to the `FIN_WAIT_2` state, it still keeps sending TCP Keepalive packets to the hypervisor. This behavior is abnormal because in this state, the TCP server should not send any further packets. After a timeout interval, the server should automatically terminate its connection, even though it did not receive the `FIN` packet from the other end.

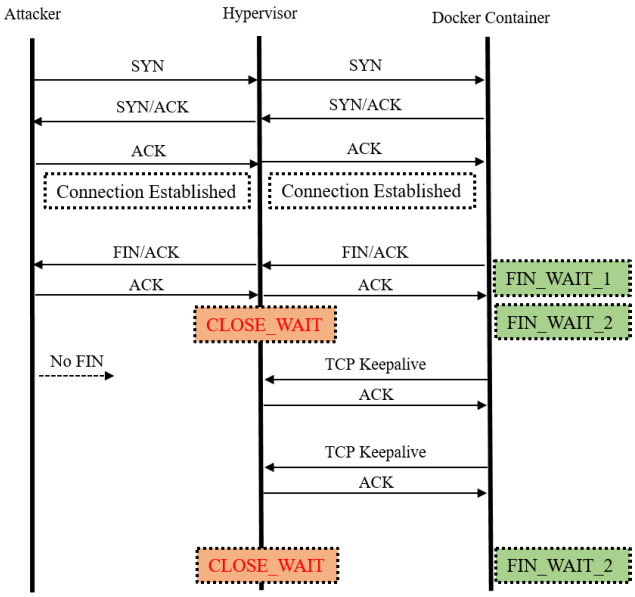

Fig. 5 Maintaining socket open in CLOSE_WAIT state with Docker container.

We assume that the TCP server operated by Docker nginx image has a setting that keeps sending probe packets to query the hypervisor about the `FIN` packet so that it could finish the handshake. Since the hypervisor did not receive any `FIN` packet from the attacker's virtual machine, it could only send an `ACK` to respond to the Docker container's query. This keeps the port of the occupied hypervisor in `CLOSE_WAIT` state for a long time, while the port from the Docker container is also stuck in `FIN_WAIT_2` state. After 24 hours, we observed that both ends were still maintaining in that state.

4. Attack Experiments

4.1 Experimental Environment

We conducted experimental tests on the recent version of VMware Workstation Pro v16.1.2. We performed all 4 attack methods presented in Chapter 3 to evaluate the feasibility of each attack on the recent hosted hypervisor. We chose VMware Workstation Pro because it is a well-known hosted hypervisor and it has

Table 3 Experimental attack results on VMware Workstation 16.1.2.

SPE Method	Port Exhaustion?	SYN Packets Sent	Keepalive Packets Sent (packets/min)	Keepalive Packets Received (packets/min)	Drawbacks
keepalive	Yes	4,700	15,200	0	Must send keepalive packets
closewait	No	4,700	0	0	Failed because of timeout (180 s)
simpleiot	Yes	4,700	0	0	Must find appropriate hosts beforehand
container	Yes	4,700	0	15,200	Receive keepalive packets

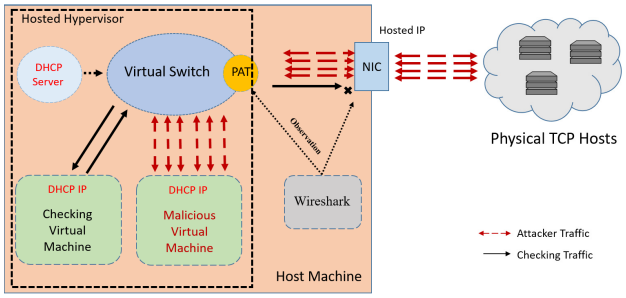

Fig. 6 Experimental model.

the NAT mode virtual switch for network communication [19].

Figure 6 shows our experimental model. We set up a virtual machine running on Linux Ubuntu 18.04 OS as the attacker's compromised virtual machine. We also set up a different virtual machine running on Windows 10 OS as a checking virtual machine. We used *iperf3*, a tool for active measurements of the maximum achievable bandwidth on IP networks [36], on this virtual machine to check the connection status of the virtual network. We also used Wireshark [38] to capture the packet transfer in both the virtual switch and the host machine's network interface card. Our host machine is a modern mid-range PC with an Intel Core i7-4790 core processor running at 3.60 GHz. The network interface is a 1 Gbps Ethernet adapter attached to the PCI-E bus. The host used Windows 8.1 64 bit as its operating system.

4.2 Attacks Execution

As mentioned in Section 3.1, we only established a constant number of TCP connections for each wave. Since a `SYN` packet is 60 Bytes, we chose to establish 15 connections each second so that we could create an attack stream lower than 1 KB/s. After the port exhaustion state, the attacker still keeps sending 15 new SYN packets per minute for 1 hour to ensure the attack would succeed.

For the keepalive-type, we used normal TCP hosts in our subnet supporting TCP keepalive feature. As described in Section 3.3.1, for each established connection, the attacker sends a TCP keepalive packet to the server after 15 seconds since the last ACK was received in order to keep the socket open. For the closewait-type attack, we chose normal TCP hosts that can perform active close and sends their `FIN` packet to the hypervisor.

For the simpleIoT-type attack, we establish connections with the abnormal TCP services that we found in Section 3.3.2. For the container-type attack, we used Docker nginx image to create 15 containers running http TCP server on each container.

For each SPE method, we conducted our attack experiment for 24 hours since we estimate that period would cause serious damage to the targeted system. We also setup an IPS program called Snort with default community rules [37] to verify that the sent and received packets will not be detected as a DoS attack.

4.3 Tesing Results and Analysis

First, we can observe that the maximum number of ephemeral ports that VMware Workstation can use for communication has been upgraded from 3,300 in version 14.1.2 [17] to 3,800 in this recent version 16.1.2. **Table 3** summarizes the result from our experiments. The keepalive, simpleIoT, and container-type methods succeed in occupying all of the available ports after 254 seconds and maintained that state until we stopped the experiments. Because the checking virtual machine could not establish a new connection to the external network, we consider these 3 methods as able to reach the port-exhaustion state.

The closewait-type on the other hand, failed to reach the port-exhaustion state. As we observed through TCPView, the hypervisor terminated the `CLOSE_WAIT` state after a timeout of 180 seconds. Therefore, we could conclude that the closewait-type method is not feasible anymore in the recent version of VMware Workstation Pro. However, by combining the closewait-type method with Docker container TCP servers, the attacker could avoid the `CLOSE_WAIT` timeout and successfully maintain the TCP connections as shown in the container-type attack results.

Among the 3 successful attack methods, the simpleIoT-type method has a lot of advantages over the other two methods. The attacker only needs to slowly send 7,600 packets within 254 seconds to establish the connections and achieve the port-exhaustion state. On the other hand, the keepalive-type must continuously sends a total of 3,800 keepalive packets every 15 seconds after connecting to the TCP hosts to maintain the connection. The container-type method does not have to send any packet after establishing the connections but still has to receive keepalive packets from the Docker containers and therefore might lose the stealth element of this attack.

5. Discussion

5.1 Limitations

Under the default community rules of Snort, the sent and received packets were not detected as DoS attacks. It is also hard to initially create a rule to identify these packets since the sent and received packets are basically normal TCP packets. We tried to implement a rule to alert the system if there are any incoming and outgoing keepalive packet, which are ACK packets not holding any data. As a result, this rule also alerted the systems when any legitimate keepalive packet is transmitted. However, the TCP hosts used for the keepalive-type and container-type can be easily detected when the system administrator investigates the traffic after port exhaustion state.

Although the simpleIoT-type is superior, the attacker has to find the appropriate IoT devices. If such IoT devices are not

found, the attack cannot be executed. However, the attacker can still use the container-type or keepalive-type method but those methods can be detected as mentioned above.

5.2 Countermeasures

Although the two newly proposed attack methods are superior to the old methods, these still have to rely on the abnormal physical IoT devices and Docker containers. Therfore, an active countermeasure against simpleIoT-type attack is to check whether there are too many `ESTABLISHED` TCP connections that have the same remote IP address in the PAT table. If there are such connections, the host machine can close the corresponding ports and block the TCP host causing the abnormal connections. The same countermeasure can be used for the container-type attack, but this time the abnormal `CLOSE_WAIT` connections should be terminated. On the newest version of VMware Workstation Pro (v.16.2.3), the `CLOSE_WAIT` connections are terminated after a timeout interval even while receiving keepalive packets from the container. This proves that the container-type method will be failed in a strict setting of the virtual switch.

Another countermeasure against the simpleIoT-type attack is that the hosted hypervisor should actively perform the close handshake after a timeout interval. For the keepalive-type and the container-type attack, an upbound number of keepalive packets that the hypervisor sends or receives per connection should be set. If the number of TCP keepalive packets in a TCP session reaches the upbound number, the hosted hypervisor must actively terminate that connection and close the corresponding port.

5.3 Ethical Consideration

We were careful to execute the experimental attacks within the established legal and ethical boundaries. In particular, we only executed the attack on our customed virtual network created from our local machine. In addition, we slowly sent a small amount of harmless traffic to the entrusted TCP hosts used for research activities in our local network. As a result, our experiments are harmless to other users and the Internet.

We have also reported the potential threat to VMware Inc. and suggested some changes in their virtual switch configuration in order to defend against future attacks in May 2021. In January 2022, we confirmed that the updated version of VMware Workstation Pro (v16.2.2) could prevent container-type method attacks. We contacted the vendor against about the indefinitely ESTABLISHED TCP state that could lead to simpleIoT-type attack. In Febuary 2022, the vendor answered that they do not consider this state as an exploitable issue according to RFC 2663. However, they will review our observations and suggestions.

On the other hand, the requirements for the simpleIoT-type method is very hard to archive since the attacker has to search for IoT devices that able to maintain such connections. Therefore, we believe the impact from disclosing this abnormal state is minor.

6. Conclusions

Because edge devices in the fog nodes might use hosted hypervisor to virtualize their resources, an adversary could use SPE attacks to sabotage the virtual switch of the hosted hypervisor and make the edge device's application unavailable to end-users. In this paper, we introduce two new SPE methods to address the drawbacks of our old methods. We also perform experimental attacks on a recent hosted hypervisor that implements NAT mode virtual switch for network connection. Testing results showed that our new SPE methods succeed in reaching the port-exhaustion state and corrupt the targeted virtual switch.

We consider that the novel SPE attack scenarios along with the old methods might be feasible for any intermediary gateway that shares the same method of handling TCP connections as the NAT mode virtual switches. Future work will revise the nature of this attack and investigate the feasibility of this attack on VPN servers or proxy servers.

Acknowledgments This work was supported by the NEC C&C Foundation Grants for Non-Japanese Researchers.

References

[1] Chen, S., Xu, H., Liu, D., Hu, B. and Wang, H.: A Vision of IoT: Applications, Challenges, and Opportunities With China Perspective, *IEEE Internet of Things Journal*, Vol.1, No.4, pp.349–359 (2014).
[2] River, W.: Virtualiztion and the Internet of Things (online), available from ⟨https://www.windriver.com/whitepapers/iot-virtualization/1436-IoT-Virtualization-White-Paper.pdf⟩
[3] Bonomi, F., Milito, R., Zhu, J. and Addepalli, S.: Fog computing and its role in the internet of things, *Proc. 1st edition of the MCC Workshop on Mobile Cloud Computing* (2012) (online), available from ⟨https://dl.acm.org/doi/pdf/10.1145/2342509.2342513⟩
[4] European Telecommunications Standards Institute: Multi-access Edge Computing (MEC) (online), available from ⟨https://www.etsi.org/technologies/multi-access-edge-computing⟩
[5] Satyanarayanan, M., Bahl, P., Caceres, R. and Davies, N.: The Case for VM-Based Cloudlets in Mobile Computing, *IEEE Pervasive Computing*, Vol.8, No.4, pp.14–23 (2009).
[6] Bitam, S., Mellouk, A. and Zeadally, S.: VANET-cloud: A generic cloud computing model for vehicular Ad Hoc networks, *IEEE Wireless Communications*, Vol.22, No.1, pp.96–102 (2015).
[7] Coutinho, A., Greve, F. and Prazeres, C.: An Architecture for Fog Computing Emulation, *Proc. WCGA* (2017) (online), available from ⟨https://sol.sbc.org.br/index.php/wcga/article/download/2552/2514/⟩
[8] Marin-Tordera, E., Masip-Bruin, X., García-Almiñana, J., Jukan, A., Ren, G.-J. and Zhu, J.: Do we all really know what a fog node is?, *Computer Communications*, Vol.109, pp.117–130 (2017).
[9] VMware ESXi, available from ⟨https://www.vmware.com/products/esxi-and-esx.html⟩
[10] Oracle Virtual Box, available from ⟨https://www.virtualbox.org/⟩
[11] Docker Desktop, available from ⟨https://www.docker.com/products/docker-desktop⟩
[12] Somani, G., Gaur, M.S., Sanghi, D. and Conti, M.: DDoS attacks in cloud computing: Collateral damage to non-targets, *Computer Networks*, Vol.109, pp.157–171 (2016).
[13] Ristenpart, T., Tromer, E., Shacham, H. and Savage, S.: Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds, *ACM Conference on Computer and Communications Security 2009* (*CCS 2009*) (Nov. 2009) (online), available from ⟨https://hovav.net/ucsd/dist/cloudsec.pdf⟩
[14] Shea, R. and Liu, J.: Understanding the Impact of Denial of Service Attacks on Virtual Machines, *20th IEEE International Workshop on Quality of Service* (*IWQoS 2012*) (June 2012).
[15] Kührer, M., Hupperich, T., Rossow, C. and Holz, T.: Exit from Hell? Reducing the Impact of Amplification DDoS Attacks, *23rd USENIX Security Symposium* (*USENIX Security 14*) (2014) (online), available from ⟨https://www.usenix.org/node/184412⟩
[16] Cambiaso, E., Papaleo, G. and Aiello, M.: Slowcomm: Design, development and performance evaluation of a new slow DoS attack, *Journal of Information Security and Applications*, Vol.35, pp.23–31 (2017) (online), available from ⟨https://www.sciencedirect.com/science/article/pii/S2214212616300680⟩.
[17] Nguyen, S.D., Mimura, M. and Tanaka, H.: Slow-port-exhaustion DoS Attack on Virtual Network Using Port Address Translation, *Proc. 6th International Symposium on Computing and Networking* (*CANDAR18*), pp.126–132 (2018).

[18] Nguyen, S.D., Mimura, M. and Tanaka, H.: SPEChecker: Checking the feasibility of Slow-port-exhaustion attack on various hypervisors, *Internet of Things*, Vol.15, 100421, ISSN 2542-6605 (2021).

[19] VMware: Workstation for Windows, available from ⟨https://www.vmware.com/products/workstation⟩

[20] Popek, G.J. and Goldberg, R.P.: Formal requirements for virtualizable third generation architectures, *Magazine Comm. ACM*, Vol.17, No.7, pp.412–421 (1974).

[21] Parallels Desktop for Mac, available from ⟨https://www.parallels.com/products/desktop/⟩

[22] QEMU, available from ⟨https://www.qemu.org/⟩

[23] Srisuresh, P. and Holdrege, M.: IP Network Address Translator (NAT) Terminology and Considerations, RFC2663 (1999), available from ⟨https://tools.ietf.org/html/rfc1999⟩.

[24] Cotton, M., Eggert, L., Touch, J., Westerlund, M. and Cheshire, S.: Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry, RFC6335 (2011), available from ⟨https://tools.ietf.org/html/rfc6335⟩.

[25] Microsoft: TCP/IP Configuration Parameters (online), available from ⟨https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc739819(v=ws.10)⟩

[26] Nguyen, S.D., Mimura, M. and Tanaka, H.: Abusing TCP retransmission for DoS Attack inside virtual network, *Information Security Applications, WISA 2017*, Lecture Notes in Computer Science, Vol.10763, pp.367–386, Springer (2017).

[27] Nguyen, S.D., Mimura, M. and Tanaka, H.: Leverage Man-in-the-middle DoS Attack with Internal TCP Retransmissions in Virtual Network, Shyamasundar, R., Singh, V. and Vaidya, J. (Eds.), *Information Systems Security, ICISS 2017*, Lecture Notes in Computer Science, Vol.10717, pp.199–211, Springer (2017).

[28] Louis, J.C. and Lee, R.E.: Introduction to Sockstress, a TCP Socket Stress Testing Framework, SEC-T Security Conference (2011) (online), available from ⟨http://insecure.org/stf/tcpdos/outpost24-sect-sockstress.pdf⟩

[29] Sieklis, B., Macfarlane, R. and Buchanan, W.J.: Evaluation of TFTP DDoS amplification attack, *Computers & Security*, Vol.57, pp.67–92 (2016) (online), available from ⟨https://www.sciencedirect.com/science/article/pii/S0167404815001285⟩.

[30] Veriato: Insider Threat Report 2018, A Veriato Whitepaper (2018), available from ⟨https://www.veriato.com/resources/whitepapers/insider-threat-report-2018⟩ (accessed 2021-12-17).

[31] Braden, R.: Requirements for Internet Hosts – Communication Layers, RFC1122 (1989), available from ⟨https://tools.ietf.org/html/rfc1122⟩.

[32] IETF: Transmission Control Protocol, DARPA Internet Program Protocol Specification RFC793 (1981), available from ⟨https://tools.ietf.org/html/rfc793⟩

[33] TCPView v3.05, available from ⟨https://docs.microsoft.com/enus/sysinternals/downloads/tcpview⟩

[34] What is a container?, available from ⟨https://www.docker.com/resources/what-container⟩

[35] nginx Docker office image, available from ⟨https://hub.docker.com/_/nginx⟩

[36] iPerf - The ultimate speed test tool for TCP, UDP and SCTP, iperf v3.1.3, available from ⟨https://iperf.fr/⟩

[37] Snort - Network Intrusion Detection & Prevention System, available from ⟨https://www.snort.org/⟩

[38] Wireshark: Network protocol analyzer, available from ⟨https://www.wireshark.org/⟩

[39] Zenmap - Official Nmap Security Scanner GUI, available from ⟨https://nmap.org/zenmap/⟩

Son Duc Nguyen received his B.E. and M.E. degrees in Engineering from National Defense Academy of Japan in 2017 and 2019. Currently, he is pursuing his Ph.D. degree in Engineering at the Department of Computer Science, National Defense Academy of Japan. He is a member of the People's Army of Vietnam since 2012. His research interests focus on low resources Denial of Service attack on virtual network environment. Mr. Son awards and honors include The NEC C&C Foundation's Grants for Non-Japanese Researchers and CANDAR2019's Outstanding Paper Award.

Mamoru Mimura received his B.E. and M.E. in Engineering from National Defense Academy of Japan, in 2001 and 2008 respectively. He received his Ph.D. in Informatics from the Institute of Information Security in 2011 and M.B.A. from Hosei University in 2014. During 2001–2017, he was a member of the Japan Maritime Self Defense Force. During 2011–2013, he was with the National Information Security Center. Since 2014, he has been a researcher at the Institute of Information Security. Since 2015, he has been with the National Center of Incident readiness and Strategy for Cybersecurity. Currently, he is an Associate Professor in the Department of C.S., National Defense Academy of Japan.

Hidema Tanaka received his B.E., M.E., and Ph.D. all in Electrical Engineering from Science University of Tokyo, in 1995, 1997, and 2000 respectively. He was a director of Security Fundamentals Laboratory at the National Institute of Information and Communications Technology until 2011. Currently, he is a Professor in the Department of C.S., National Defense Academy of Japan.