Regular Paper

SVTester: Finding DoS Vulnerabilities of Virtual Switches

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

Received: December 1, 2020, Accepted: June 7, 2021

Abstract: Nowadays, virtualization is being deployed in many companies and institutions' systems. However, a noticeable security problem of virtualization is the fact that multiple virtual machines are run on one physical host machine called *hypervisor*. Hypervisors often implement a virtual switch to manage network connections between the internal virtual network and the external physical network. However, an adversary could exploit virtual switch flaws and use them to sabotage the entire virtual network. As a consequence, the attack could make all applications running on virtual machines unavailable. In this paper, we present SVTester, a fuzzing-based testing tool that can automatically identify possible vulnerabilities of a virtual switch that can be exploited for certain types of Denial-of-Service attack. We used an initial version of SVTester to check several hypervisors that implement the virtual switch. The results show that SVTester was able to rediscover DoS weaknesses on an old version of VMware hypervisor and found a novel possible vulnerability in the Oracle VirtualBox hypervisor. Our results also prove the effectiveness and potential of SVTester in evaluating virtual network security.

Keywords: virtual network, security testing, DoS attack

1. Introduction

As computer networking technologies continue to evolve, network security has an important role in today's networked world. However, improper implementation of network applications and the complexity of many network protocols make errors inevitable. The errors can become vulnerabilities of the network system and, in consequence, attackers can exploit those vulnerabilities and deploy severe attacks. The cost to recover the damage from attacks can be very high. For example, a survey from Corero shows that Denial-of-Service (DoS) attacks can cost enterprises up to $50,000 and also the loss of their customer trust and confidence [6]. Therefore, identifying possible bugs and vulnerabilities of a network's system before zero-day attacks is essential.

DoS attack is one of the most intimidating attacks on any network system. A typical method to initiate a DoS attack is sending overwhelming amounts of seemingly legitimate traffic to the target and flood the target's network layer [17]. When the attacker employs multiple infected systems which are often part of a botnet, the attack is called Distributed Denial of Service (DDoS) [5]. With the development of network technologies, DDoS attacks are growing in scale and frequency each year. Recently, in February 2020, the largest DDoS attack in terms of traffic to date has been launched against Amazon Web Services, with a peak flow of 2.3 terabits per second (Tbps). Another famous DoS attack with 1.3 Tbps was recorded in 2018 against GitHub, a popular online code management service [3]. Furthermore, with other approaches in attack methods such as Slow DoS Attack, DoS attacks have also become more sophisticated and harder to detect [9].

DoS attacks can also cause serious damage to any virtual network system. Shea and Liu prove that virtual networks are more vulnerable under TCP SYN DoS attacks [18]. In particular, a light DoS attack can lead to a 50% decrease in the virtual server's performance when compared to the non-virtualized system using the same amount of resources. In 2016, Somani et al. show more results of the effect of DoS attack on cloud computing [10]. Their research shows that when a virtual machine (VM) is targeted with a DoS attack, other non-targeted VMs and servers sharing the same host machine are also affected. They also demonstrate that some features of cloud computing such as resource race and autoscaling increase the effect of DoS attacks on cloud infrastructure.

The decline of DNS and NTP Amplification DDoS attacks proves that proper understanding of exploitable vulnerabilities can help in mitigating DoS attacks [16]. Therefore, it is important to have effective methods and tools to discover new possible vulnerabilities that can lead to a new DoS attack. One of the well-known approaches for testing tools is *fuzzing*. Fuzzing techniques can be various, but they share the same advantages of simplicity and effectiveness over other complex testing approaches.

This paper is a continuation of our works presented in Refs. [22] and [23]. In the first work, we proved that some behaviors of the virtual switch could become possible vulnerabilities that might be exploited for DoS attacks [22]. In the following work, we introduced different DoS attack scenarios abusing those possible vulnerabilities [23]. However, our previous works only focused on one susceptible hypervisor, which is VMware Workstation Pro 12.0.0. This leads to the fact that other hypervisors might also have the same possible vulnerabilities. Therefore, we need to further investigate the TCP retransmission behaviors of various hypervisors.

[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

In this paper, we introduce a network testing tool called SVTester, which can be used for testing intermediary servers and network software. Motivated by the general idea of fuzzing and Kuhrer method in exploiting TCP three-way-handshake for amplification DoS attack in the physical network [16], SVTester can automatically generate and monitor a half-open TCP session between a client and a TCP host. In this paper, we used SVTester to investigate retransmission flaws of virtual switches that can be exploited for certain types of TCP DoS attacks. Our testing result on VMware 12.0.0 shows that SVTester was able to automatically rediscover 2 possible vulnerabilities found in the previous research [22]. Furthermore, SVTester found a new possible vulnerability in a recent version of the Oracle VirtualBox hypervisor.

In summary, we make the following contributions:

(1) We consider Kuhrer method of abusing TCP three-way-handshake and design a testing tool that can test the behavior of retransmitting TCP packets from different hypervisors.
(2) We perform testings and evaluate the security of various hypervisors.
(3) We discover a novel possible vulnerability in the VirtualBox hypervisor that can be abused for DoS attacks.
(4) We propose DoS attack schemes and evaluate the impact of the attacks on the targeted hypervisors.

This paper is organized as follows: Section 2 presents the background information of this research. We review the related works in Section 3. Section 4 explains our testing method and the algorithm of SVTester. We show our testing experiments and results in Section 5. Section 6 describes our DoS attack schemes based on the possible vulnerabilities found and also analyzes our experimental attacks. Section 7 evaluates the security of various hypervisors and also discusses the limitation of our methods. Finally, Section 8 concludes and discusses future works.

2. Background

2.1 Virtualization

Virtualization is a technology that helps users create a software-based representation of hardware platforms, servers, storage, or computer network. This technology has changed the network industry through its powerful capability in balancing demands for resources. Recently, the number of newly installed virtual systems has surpassed the number of newly installed physical servers. A recent survey showed that 90% of organizations use virtualization in their IT infrastructures and a further 34% of organizations are using virtual servers to meet the majority of server needs [2], [8]. Virtualization technology has also opened the gate to virtual private network (VPN), software-defined network (SDN), and cloud computing. One well-known example in the industry is Cloud Computing Amazon EC2, which provides customers with secure, resizable computational resources [29].

One of the main components of virtualization is virtual machine manager (VMM), also known as *hypervisor*. A hypervisor is computer software, firmware, or hardware that enables virtualization. The hypervisor operates one or more virtual machines on a physical computer called *host machine* while each virtual machine is called a guest machine. The guest operating systems often require special drivers or to be specifically designed to be run on the hypervisor. However, in some cases, an operating system can run native and unaltered on the hypervisor necessarily knowing it is virtualized [4]. The hypervisor provides the guest systems a virtual operating platform and supervises the execution of the guest systems. The hypervisor also makes sure resources are allocated to the guests as necessary. In 1974, Popek and Goldberg classified hypervisors into two types [11]:

Type-1, native or bare-metal hypervisors: These hypervisors run directly on the host's hardware to control the hardware and to handle VM's operating systems. It is responsible for the allocation of all resources, such as disk, memory, and CPU, to its VM. These hypervisors only require a small footprint and occasionally, they have very limited driver databases limiting the hardware on which they can be installed. Some also need a privileged VM, known as a Domain-0 or Dom0, to provide access to the management and control interface to the hypervisor itself. The Type 1 hypervisor is typically being used in the server virtualization environment. Modern examples include Xen, XCP-ng, Oracle VM Server for x86, Microsoft Hyper-V, Xbox One system software, and VMware ESX/ESXi.

Type-2, hosted hypervisors: These hypervisors operate on a common operating system just as other computer programs do. In other words, it is installed on top of the host operating system. Type-2 hypervisors abstract guest operating systems from the host operating system and make the guest system run as a process on the host. Therefore, it typically has fewer hardware/driver issues because the host operating system is responsible for interfacing with the hardware. However, the additional overhead can cause a hit on performance compared to Type 1 hypervisors. Type-2 hypervisors are often being used in small companies and workgroups due to their convenience in creating and managing virtual machines without an additional management console. Some examples of this type are VMware Workstation, Oracle VirtualBox, Parallels Desktop for Mac, and QEMU.

The hypervisor uses a virtual switch to control all VM's Ethernet connections. The virtual switch is a software construct that performs Ethernet frame switching functionality and runs within the active memory of the hypervisor. The virtual switch can employ one or many physical network adapters to communicate with other computers in the physical network. It can provide several virtual network adapters, which are also software constructs that are responsible for receiving and transmitting Ethernet frames into and out of their assigned VM. If an attacker sabotages the virtual switch with a DoS attack, the entire virtual network will be broken down.

The virtual switch often has 2 options to allow network connections from a VM to the external physical network. They are called Bridged mode and NAT mode. Bridge mode is the option when the virtual switch directly connects the VM to the physical network and allows the VM to get its IP address from a physical DHCP server. On the other hand, NAT mode is the option when the virtual switch uses Port Address Translation (PAT) to assign the VM's IP address from a virtual DHCP server. **Figure 1** illustrates a virtual switch in NAT mode allowing connections between VMs and an external physical network. In this mode, when a VM sends a packet to a server on the external network, the

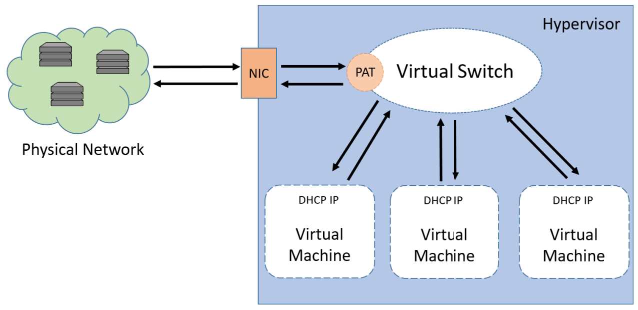

Fig. 1 Virtual switch in NAT mode.

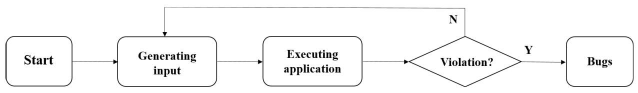

Fig. 2 An example of a fuzzing process.

virtual switch translates the VM's IP to the hypervisor's IP and assigns a port number used for the transmission. The hypervisor then employs a dynamic port number assigned by the virtual switch to send a new packet to the physical server. With that transmission process, a NAT mode virtual switch can be considered as an intermediary server between the VM and the external servers [24].

The fact that multiple VMs are operated on one physical host machine can create security issues. In 2009, Ristenpart et al. introduced a threat scenario when the customer's VM is assigned to the same hypervisor as their adversary [25]. According to their research on Amazon EC2, the attackers can check if their VM is on the same hypervisor with the target's VM. Furthermore, the attackers can create and delete their VM multiple times until the adversary VM is operated on the same hypervisor with their target. After that, the attackers can proceed to attack the target VM with side-channel or DoS attacks.

2.2 Fuzzing

Fuzzing is a famous automated black-box approach to the security testing and vulnerability finding of applications. During the fuzzing process, the tested system is provided with random or unexpected input, which can trigger some flaws contained in the application [7], [13], [26]. A fuzzing process often has four main steps: generating random or unexpected data that could lead the tested application into an invalid state; executing the application with the generated input; observing the output caused by the generated input; and finally analyzing the exception. **Figure 2** indicates a general example of a fuzzing process.

A fuzzing process starts with the generation of inputs, which are also called testcases. There are two different strategies for creating inputs. The *generation* strategy generates a set of valid input values according to a configuration file that predefines the file format. The inputs are then modified with fuzzing primitives to obtain faulty input data. On the other hand, the *mutation* strategy extracts a set of valid input values from the normal sessions and modifies the inputs with fuzzing primitives [12].

The inputs are fed to the target application after generated in the previous step. Fuzzer automatically executes the target application and observes all outputs. When violations such as crashes or abnormal behaviors are captured, fuzzer stores the input for later analysis. In some cases, fuzzer also needs to perform several intermediate steps in order to bring the system to a certain state that could expose flaws of the system.

Fuzzing's greatest advantage is the fact that it only relies on the input and output of the tested application without requiring any knowledge of its internal structure. Therefore, if the tested system is completely closed, fuzzing is the only method to check that system's quality. Even though it is not expected to expose all faults in the tested application, bugs found through fuzzing are guaranteed to correspond to some flaws in the tested application's source code.

Hypervisors fuzzing has some challenges because it requires interacting with different devices using various protocols and interfaces. Therefore, there appears to be a lack of research on fuzzing hypervisors. One example of a state-of-the-art hypervisor fuzzing tool is HYPER-CUBE [19]. This fuzzer has to rely on a fully custom OS called HYPER-CUBE OS booting inside the target hypervisor. This specified OS can give the authors full control over the fuzzing process. As a consequence, this fuzzer has greater performance than previous coverage-guided fuzzers such as VDF [1]. Another hypervisor fuzzer example is NYX, which uses coverage-guided fuzzing with fast snapshots and affine types to accelerate the fuzzing process [20]. The drawback of NYX is that it is slightly more complex to set up than HYPER-CUBE, as the target hypervisor needs to run inside a modified hypervisor called KVM-PT.

In our research, we only focus on the behavior of the virtual switch in handling TCP retransmission. Therefore, we follow a simpler approach than state-of-the-art hypervisor fuzzing methods. Our approach shares some elements of *network protocol fuzzing*, which is to send malformed data to the target through socket APIs and observe the exceptions. One example of a testing tool that borrows the notion of network protocol fuzzing is DELTA [21]. DELTA is a SDN penetration tool that randomizes the control flow sequence and the priority data in the `FLOW_MOD` message of the OpenFlow protocol. With the malformed packets, DELTA was able to reproduce many attack scenarios and also discover new vulnerabilities on different SDN controllers. However, our testing tool has a different approach to DELTA. Instead of utilizing malformed packets, we aim to create a disruption in TCP handshake that could expose the behavior of the virtual switch in an abnormal situation.

3. Related Work

3.1 TCP Amplification Attack

Among DoS attack techniques, abusing UDP-based public servers like DNS or NTP for reflective amplification attacks is commonly being used. In this attack, the attacker sends comparatively small requests with the victim's source IP address to the hosts that reflect significantly larger responses to the victim. These hosts are defined *amplifiers* [16]. For example, the attacker, which is spoofing the victim's IP address, generates and sends 1 Mbps of DNS queries to an exploitable DNS server. The DNS server then returns 100 Mbps of traffic to the victim. This makes the victim's servers become busy handling the attack traffic. As a consequence, the victim server cannot service any other request from legitimate users and the attacker achieves a denial-of-service.

On the other hand, TCP-based public servers are not known to be abused for DDoS amplification attacks [5]. Comparing with UDP, the attacker apparently cannot use TCP-based public servers as amplifiers because of the three-way-handshake sequence. If the attacker spoofed the victim's IP address, the victim would not acknowledge an unknown SYN/ACK packet. Subsequently, the three-way-handshake would not complete and the TCP/IP framework does not allow payload data for amplification attacks.

However, Kührer et al. show that the attacker can still abuse the TCP retransmission function during the handshake for a reflective amplification DoS attack [15]. When the client does not send any responses after receiving a SYN/ACK packet from the server, the server will consider that the SYN/ACK packet has been lost and will resend the SYN/ACK packet again in a timeout interval until it receives any responses. The retransmission of SYN/ACK can make an amplification situation as UDP-based servers. Furthermore, because the TCP handshake is not completed, the SYN/ACK packets can be reflected to the victim by IP spoofing techniques. In fact, thousands of TCP-based public servers that can be abused for amplification attack as amplifiers have been found [16].

Motivated by Kuhrer's findings on physical TCP hosts, we want to evaluate the potential of this attack on virtual network environments. In our previous work, we proved that the virtual switch of a hypervisor could be used as an amplifier because of its TCP retransmitting behavior [22]. The following section reviews our finding of virtual switch's behaviors on retransmitting TCP packet.

3.2 Virtual Switch's TCP-retransmitting Behaviors

As mentioned in Section 2.1, the virtual switch in NAT mode can be considered as an intermediary server between VMs and external physical hosts. Theoretically, an insider attacker could abuse the virtual switch as an amplifier for TCP Amplification Attack to disturb the other virtual machines on the same physical machine.

In our previous research, we observed TCP retransmission in VMware Workstation Pro 12.0.0 [22]. We chose VMware Workstation Pro for the following two reasons. First, VMware Workstation Pro is the industry standard Type-2 hypervisor that can run multiple operating systems simultaneously on a single host machine [38]. Second, with the Shared VM feature, we can configure the host machine as a VMware Workstation Server and share VMs over the local network.

In order to make the virtual switch retransmit TCP packets, we followed Kührer method [15] to trigger the retransmission (**Fig. 3**).

- Step-1: From a virtual machine, we send a single SYN packet ((1) SYN in Fig. 3) to a TCP host in the local subnet.
- Step-2: TCP hosts reply by sending SYN/ACK packet ((2) SYN/ACK in Fig. 3).
- Step-3: In order to activate TCP retransmission in the three-way-handshake, we set that virtual machine does not reply RST packet to TCP hosts ((3) in Fig. 3).

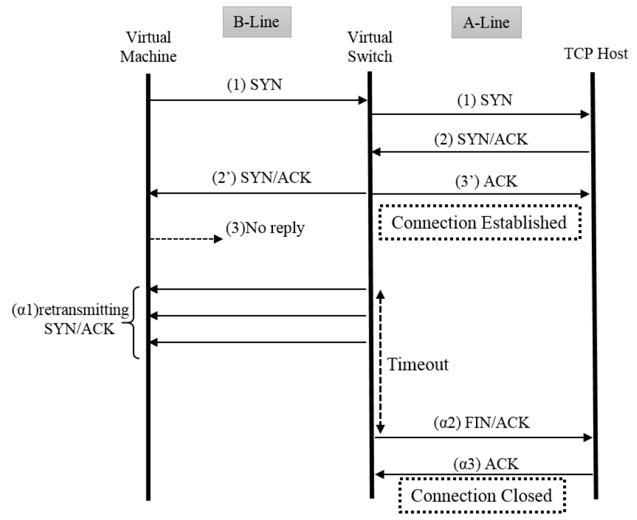

Fig. 3 Observing TCP retransmission from a virtual switch.

We observed communication situations in external ("A-line" in Fig. 3) and internal ("B-line" in Fig. 3) of the virtual network using Wireshark [39] on the host PC. From some experiments, we found the following [22]:

(1) In A-line, the three-way-handshake is established between the virtual switch and the external TCP host. This three-way-handshake is normal, therefore no retransmissions are found in the external communication.

(2) In B-line, we observed many retransmitting SYN/ACK packets from the virtual switch (($\alpha 1$) in Fig. 3). This retransmission executes in a time-out interval, which is 30 seconds since the first SYN/ACK packet was sent. We deduce that this is VMware Workstation's retransmission default setting. After stopping retransmit SYN/ACK packet to the virtual machine, we observed from A-line that the virtual switch sent a FIN/ACK packet to the hosts in order to end the established TCP connection (($\alpha 2$) in Fig. 3).

(3) The retransmitting frequency from our experiments is far different from Kuhrer's observation on physical servers. In the real network, the SYN/ACK retransmitting frequency is only 5 packets in 30 seconds [15]. However, the VMware Workstation Pro 12.0.0 hypervisor retransmits 300 packets in 30 seconds, which is much larger than a physical server in a real network.

From the observation above, we theorized that the virtual switch itself could become an amplifier with a high amplification factor for reflective DoS attacks targeting the internal virtual network. However, our previous research was limited to VMware Workstation Pro 12.0.0. Therefore, in this work, we intend to expand the investigation to various hypervisors. For that purpose, we designed an automated testing tool that could generate a half-open TCP session and analyze the retransmission from the targeted virtual switch. From the retransmission behaviors observed, our tool can identify possible vulnerabilities of the targeted virtual switch that could be exploited for DoS attacks.

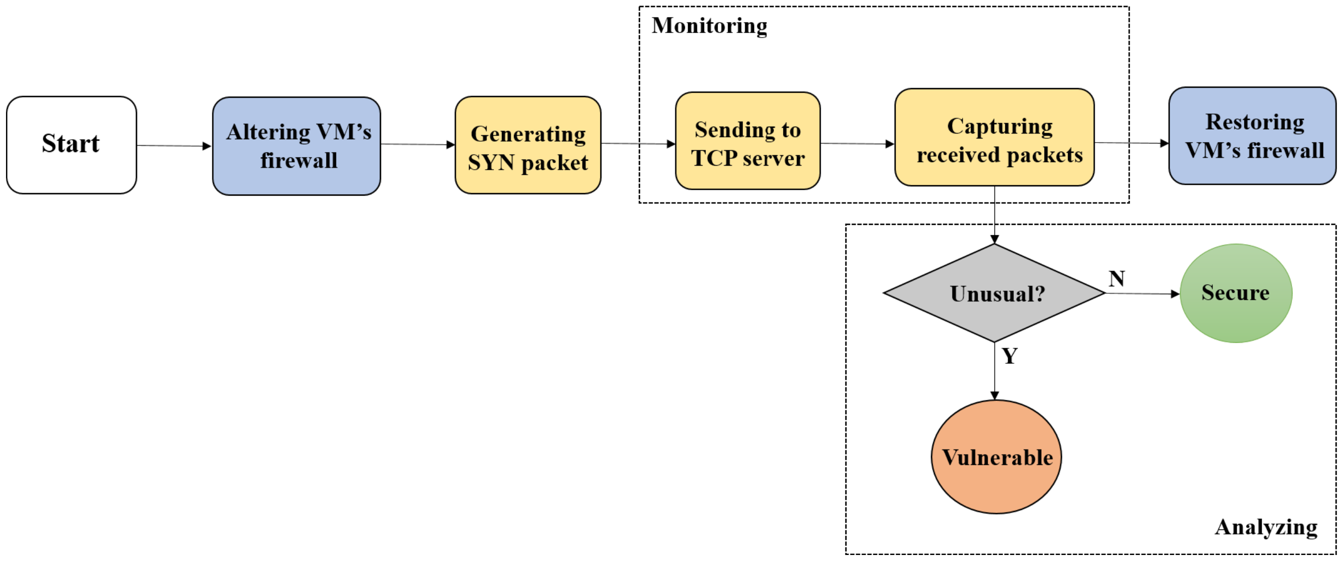

Fig. 4 Working process of SVTester.

4. SVTester Description

4.1 Working Process in General

In this section, we describe the process and algorithm of our testing tool, which is called SVTester. In the current implementation, SVTester can operate on any VM running Linux OS. SVTester's working process consists of 4 main stages, which are: altering VM's firewall, generating TCP packet input, monitoring the TCP transmission, and analyzing the result. **Figure 4** shows the working process of SVTester.

The principle of SVTester is related to the method of triggering TCP retransmission mentioned in Section 3.2. To summarize, SVTester initiates a TCP handshake with an external TCP server but does not respond to the incoming `SYN/ACK` packet. Normally, if there are no TCP listening sockets for a specified port, any TCP packet coming for that port will trigger a `RST` packet from the OS kernel. To prevent this event, in the first stage, SVTester alters the tested VM's firewall by blocking all outgoing `RST` packets.

In the second stage, SVTester generates a raw TCP `SYN` packet from the Linux socket. This `SYN` packet's structure follows the standard TCP protocol specification [28]. After the generation, SVTester sends the `SYN` packet to a specified TCP server connecting to the hypervisor. At stage 3, SVTester monitors and captures all incoming packets from the TCP server. In the final stage, SVTester checks for violations and reports any abnormal behaviors of the virtual switch based on the number of retransmitted packets and the timeout of the TCP session. Finally, SVTester restores the VM's firewall to its previous settings.

4.2 Assigning Unusual Thresholds

In our initial version of SVTester, we assign two unusual thresholds of a TCP session. The first unusual threshold is decided by the number of retransmitted packets (*RP*) measured in 60 seconds counting from the start of the fuzzing process. If *RP* exceeds a specific value, then we determine that the retransmission is abnormal. Since RFC793 and RFC1122 do not specifically mention the upper bound value of `SYN/ACK` retries, we used a suggestion in the Linux TCP manual as a reference to specify our threshold. According to the manual, the number of retransmitted `SYN/ACK` should not be higher than 255 [27]. Therefore, we design that if the number of *RP* is larger than 255, the tested hypervisor is reported having an unusual threshold in TCP retransmission.

The second unusual threshold is decided by the timestamp of the last incoming TCP packet. According to Ref. [28], the default timeout for TCP retries should be 5 minutes. Therefore, we determine that if the timestamp of the last TCP packet captured from the testing TCP session is above 300 seconds, the tested hypervisor might have a possible vulnerability that could be exploited for DoS attacks.

4.3 Main Algorithm

Algorithm 1 shows more insight into the working process of SVTester. S is an attribute set for specifying the source of the `SYN` packet such as VM's IP address, source port number, and virtual network interface. D is an attribute set for specifying the destination of the `SYN` packet such as destination IP address and destination port number. T is the observation time. Since the timeout validation is 300 seconds, T should be much longer than 300 seconds so that SVTester can fully observe any retransmitted packet beyond the 300 seconds timestamp. In this paper, we chose $T = 3{,}600$ seconds to observe the generated TCP sessions in 1 hour.

V is the state output of SVTester, which can suggest that whether the tested hypervisor is having unusual TCP retransmission or not. For further analysis, we design SVTester to also show the number of retransmitted packets in 60 seconds (*RP*) and the timestamp of the last TCP packet captured from this session (TS).

Variable t represents the current time while t_0 is the starting time.

To set up `RST` blocking rule for testing VM's firewall, SVTester alters the *iptables*, which is a Linux utility that allows configuring the IP packet filter rules of the Linux kernel firewall. For sniffing task, SVTester is implemented with *Scapy* module [36]. Each

Algorithm 1: SVTester algorithm

```
input  : $S$, $D$, $T$
output: $RP$, $TS$, $V$
1  modify iptables;
2  generate TCP SYN packet with $S$, $D$;
3  $t_0 = time.time()$;
4  $RP = 0$;
5  send SYN packet;
6  while $t < t_0 + T$ do
7      sniff incoming TCP packets $p$;
8      if $p\ source = D$ then
9          $TS = t - t_0$;
10         write $p$ to log file;
11         if $TS < 60$ then
12             $RP + 1$;
13         end
14     end
15 end
16 if $RP > 255$ or $TS > 300$ then
17     $V = true$
18 else
19     $V = false$
20 end
21 return $V$, $RP$, $TS$;
22 restore iptables;
```

time a TCP packet coming from the specified TCP server (D) is captured, SVTester computes TS by subtracting the starting time t_0 from the current time t.

5. Testing Experiments

5.1 Experimental Environment

We conduct testings on various well-known hypervisors that implements NAT mode virtual switch for network communication such as VMware Workstation Pro (v15.5.6) [38], Oracle VirtualBox (v6.1.12) [32], Parallels Desktop 15 [33], Hyper-V (on Windows 8.1) [31], KVM QEMU (v2.11.1) [35]. We also include testing on the susceptible VMware 12.0.0, which has possible vulnerabilities found in our previous research [22]. The reason to include testing on this hypervisor is that we want to check whether our testing tool can reproduce the possible vulnerabilities found on this hypervisor or not.

In order to test each hypervisor, we created representative test networks and systems in our subnet. We decided to perform experiments on our customized systems instead of rented resources for several reasons. The first reason is that we want to completely gain access to the hypervisor and therefore could observe all of its connections with the external network. Secondly, cloud providers such as Amazon EC2 do not permit DoS related testing on their systems. And finally, using our custom system, we could ensure the hardware resources remain constant during our experiments.

Figure 5 indicates our experimental model. We used a modern mid-range PC with an Intel Core i7-4790 core processor running at 3.60 GHz as our host PC. 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. We also used a TCP Host in our real subnet that connects to our host PC. This TCP host is actually utilized for office work and research activity.

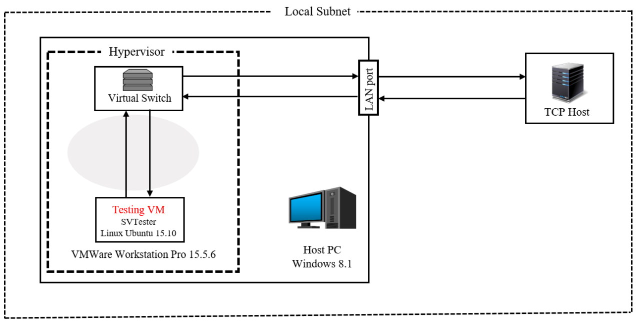

Fig. 5 Experimental model.

5.2 Implementation

For each hypervisor, we created a virtual machine running Linux Ubuntu 15.10 OS. We implemented our initial version of SVTester on this VM and use it to find the possible vulnerabilities of the operating hypervisor. This VM contained some specific Python 3 library such as *Scapy* [36] to support SVTester's working process.

Initially, we provided SVTester with the source and destination IP addresses and port numbers. SVTester then automatically executed the testing process (as mentioned in Section 4). After the testing process finished, SVTester displayed the results, which are the number of retransmitted packets, the timestamp of the last TCP transmission, and the suggestion of possible vulnerabilities that the testing hypervisor might have.

5.3 Testing Results

Table 1 shows the testing results of various hypervisors using SVTester. The testing on VMware Workstation Pro 12.0.0 shows that SVTester was able to find the possible vulnerabilities first discovered in our previous research [22]. However, the results also indicate that only VMware Workstation Pro 12.0.0 has a very high TCP retransmission frequency, which is 300 packets in 30 seconds, while all of the other hypervisors have a low TCP retransmission frequency, which only range from 4 to 7 retransmitted packets in 30 seconds.

On the other hand, the last timestamp output varies between hypervisors. Most of the hypervisors have a TCP session timeout output under 300 seconds, which is the acceptable value (mentioned in Section 4.2). As expected from VMware Workstation Pro 12.0.0, SVTester shows that the last timestamp was over 300 seconds and therefore violated the TCP protocol. This is related to the possible vulnerability found in our previous research [22]. However, we also discovered that besides VMware 12.0.0, Oracle VirtualBox 6.1.12 also has a long TCP session timeout, which was above 300 seconds. Therefore, we could determine that Oracle VirtualBox 6.1.12 hypervisor might have a novel possible vulnerability that could be exploited for DoS attacks.

From the testing result, we summarize 2 possible vulnerabilities of a virtual switch that could be exploitable by DoS attackers.

(1) $V1$ **- High TCP retransmission frequency.** The virtual switch retransmits a large number of SYN/ACK packets, which makes it become a potential amplifier for a reflective amplification DoS attack.

(2) $V2$ **- Abnormal retransmission timeout.** The virtual switch

Table 1 Fuzzing results on various hypervisors.

Tested Hypervisor	RP	Timestamp	Possible Vulnerability
VMware Workstation Pro 12.0.0	300	3,600 s	$V1$, $V2$
VMware Workstation Pro 15.5.6	7	15.3 s	
Oracle VirtualBox 6.1.12	5	633 s	$V2$
Pararells Desktop 15	6	59.8 s	
Hyper-V (Windows 8.1)	4	94.1 s	
KVM QEMU	4	46.5 s	

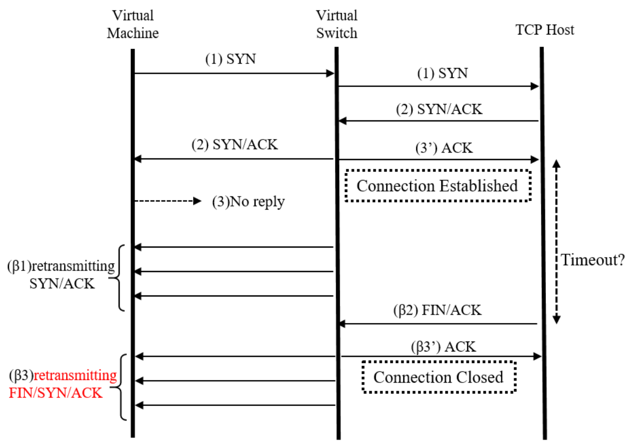

Fig. 6 Abnormal retransmission timeout case.

violates the standard timeout of retransmitting TCP packets. **Figure 6** illustrates this particular case. It happened when the TCP host sends a `FIN/ACK` packet ((β2) in Fig. 6) before the virtual switch terminates the half-open TCP connection. The virtual switch subsequently inserts the `FIN` flag to the retransmitting packet, thus makes a `SYN/ACK` packet become a `FIN/SYN/ACK` packet ((β3) in Fig. 6). This retransmission keeps on resending the packets until receiving a response from the VM and thus violates the standard timeout interval. This behavior could benefit attackers when they want to force the virtual switch to retransmit meaningless TCP packets and occupy the hypervisor's memory for a long time.

6. DoS Attack Schemes

From the possible vulnerabilities found above, we proposed 2 DoS attack schemes $S1$, $S2$, in which the attacker has a compromised virtual machine inside the targeted network and has the ability to send low-rate attack traffic. These attack schemes might be able to sabotage the entire virtual network or overload the virtual system from inside.

6.1 $S1$ - Internal TCP Retransmission Attack

6.1.1 Attack Description

With the $V1$ possible vulnerability, the attacker can perform an internal TCP retransmission attack. In this attack scheme, we assume the attacker possessed one VM in the virtual network and his purpose is to sabotage other legitimate VMs' activities. The main goal of the attacker is to make the virtual switch busy by continuously generating a lot of retransmitting TCP packets and sending those packets to the attacker's VM. As a result, this attack slows down the targeted virtual switch's performance. This attack is harder to detect than a direct flooding attack because the incoming packets come from its own virtual switch.

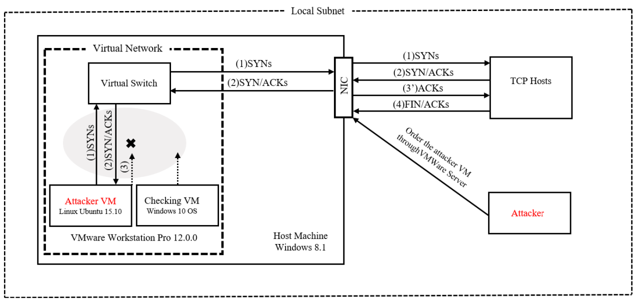

Fig. 7 $S1$ attack model.

In summary, the attack consists of three main steps.

- Step-1: The attacker sends `SYN` packets from his VM to many TCP hosts in the local subnet.
- Step-2: TCP hosts reply by sending `SYN/ACK` packets. The virtual switch will transmit the `SYN/ACK` packet to the VM and spontaneously sends back its `ACK` packets to complete the three-way-handshake between the host machine and external TCP hosts.
- Step-3: In order to trigger TCP retransmission in the three-way-handshake, the attacker makes his VM not reply `ACK` or `RST` packets to TCP hosts. As a result, the virtual switch will retransmit a large number of `SYN/ACK` packets to the attacker's VM.

6.1.2 Experimental Attack

In order to evaluate the effect of $S1$ attack on the other virtual machine operating on the same vulnerable hypervisor, we performed experimental attacks on our virtual network. **Figure 7** illustrates our experimental attack model. In this experiment, the attacker is a machine in our local subnet. This machine could gain access and control a virtual machine in the hypervisor through VMware's Shared VM feature. We also used TCP Hosts in our local subnet that connect to our host PC. These TCP hosts are actually utilized for office work and research activity and those administrations are entrusted to each section.

To monitor the change in the performance of the Checking VM under normal and attacked conditions, we chose the PassMark PerformanceTest 10.1 [34] to measure the CPU, memory, and I/O performance. To observe the network performance, we used iPerf 3.1.3 [30] to find the maximum bandwidth between the Checking VM and a TCP Host in our subnet.

In this experiment, the Attacker VM sent 100 `SYN` packet to 30 TCP Hosts, which resulted in 3000 `SYN/ACK` retransmissions. According to our testing results in Section 5.3, for each retransmission VMware Workstation 12.0 generated 10 SYN/ACK packets per second. Since one `SYN/ACK` packet is 60 bytes, our experimental attack created 14.4 Mbps of attack traffic from 180 KB of the initial `SYN` packets.

Table 2 The Checking VM's performance while Idle - Under $S1$ Attack.

Benchmark	Idle	Under Attack	Degradation
CPU Single Threaded	2,033 MOps	1,824 MOps	10.28%
Memory Read	9,033 MBps	8,576 MBps	5.06%
I/O	9 MBps	9 MBps	0%
Network Bandwidth	95.4 Mbps	92.4 Mbps	3.14%

Table 2 shows our experimental attack results. Overall, our $S1$ experimental attack slightly impacts the performance of the other VM on the same hypervisor. Under a bandwidth of 14.4 Mbps of retransmitted `SYN/ACK`, the CPU Single Threaded benchmark is decreased by 10.28% while the Memory Read benchmark also slightly reduces by 5.06%. The maximum network bandwidth is also degraded by 3.14%. However, the I/O performance was not influenced under our experimental attack.

6.2 $S2$ - Long timeout TCP Session Attack

6.2.1 Attack Description

With the $V2$ possible vulnerability, the attacker can create a large number of half-open TCP sessions with a long timeout. In this attack scheme, we assume the adversary is controlling one VM inside the targeted network. The attacker also needs to ensure that the external TCP Hosts perform an active close and send a FIN/ACK packet before the virtual switch ends the half-open TCP connections.

In summary, the attack consists of four main steps.

- Step-1: From insider's VM, the attacker sends `SYN` packets to a large number of TCP Hosts having network connection with the hypervisor.
- Step-2: The TCP Hosts subsequently reply by sending their `SYN/ACK` packets.
- Step-3: In order to create half-open TCP sessions, the attacker sets that the VM does not reply `ACK` or `RST` packets to any TCP Hosts. On the other hand, the virtual switch spontaneously completes the handshakes with all external TCP Hosts by sending its own `ACK` packets.
- Step-4: The TCP Hosts send `FIN/ACK` packets to the hypervisor and thus create long timeout TCP sessions.

6.2.2 Experimental Attack

We also performed $S2$ experimental attacks on both VMware Workstation 12.0.0 and Oracle VirtualBox 6.1.12 to evaluate the impact of the attack. The experimental model is the same as the $S1$ experimental attack in Section 6.1.2. In this experiment, the Attacker VM attempts to create 10,000 long timeout TCP sessions. We also used PassMark PerformanceTest 10.1 to measure the CPU, memory, and I/O performance of the Checking VM and used iPerf 3.1.3 to measure the maximum network bandwidth. To further observe the TCP session state, we used TCPView [37] to check the status of TCP ports used by the hypervisor.

In the $S2$ attack experiment on VMware Workstation 12.0.0, we observed only 3,400 TCP `CLOSE_WAIT` sessions despite the Attacker VM tried to create 10,000 sessions. We also observed 3400 `FIN/SYN/ACK` retransmissions with a bandwidth of 16.32 Mbps. As a result, the degradation of CPU and memory performance benchmark showed no major change compared to the result of the $S1$ attack in Table 2. The I/O performance under attack also had no change compared to the normal condition. However, the iPerf3 on the Checking VM failed to connect to the outside server. The reason is that all of the available dynamic ports used for VM connections on VMware 12.0.0 NAT were occupied. As a consequence, the Checking VM could not connect to the outside physical network for a long time. After 24 hours, the Checking VM was still disconnected to the outside host. We conclude that the attack successfully sabotaged the VMware 12.0.0 virtual network with 3400 long timeout TCP sessions.

On the other hand, the $S2$ attack experiment on Oracle VirtualBox 6.1.12 shows different results from the attack on VMware 12.0.0. We observed that 10,000 TCP `CLOSE_WAIT` sessions were created and maintained in 10 minutes. After the timeout, Oracle VirtualBox 6.1.12 cleared the `CLOSE_WAIT` sessions. However, the CPU, memory, I/O, and network benchmarks under attack show no difference with the performance in the normal condition. We conclude that a $S2$ attack with a scale of 10,000 long timeout TCP sessions could not affect the performance of the Oracle VirtualBox 6.1.12 virtual network.

6.3 Countermeasures

To counter these attacks, we suggest some changes in the fundamental settings of the virtual switch TCP retransmission to eliminate the possible vulnerabilities. First, the `SYN/ACK` retransmission frequency should be limited to under 10 packets in 30 seconds. We also performed a $S1$ attack experiment on Oracle VirtualBox 6.1.12 and the results showed no significant drop in the performance compared with the normal condition. In other words, virtual systems with strict rules for TCP retransmissions are safe against the $S1$ attack. To mitigate the $S2$ attack, the hypervisor should include a NAT timeout parameter to terminate the half-open TCP session with long timeout. In particular, we suggest that the virtual switch should send a RST packet to the virtual machine after a certain number of SYN/ACK retransmissions.

7. Discussion

7.1 Evaluating Hypervisors

From the testing results above, we could conclude that VMware Workstation Pro 15.5.6, Parallels Desktop 15, Hyper-V, and KVM QEMU hypervisors are unexploitable as amplifiers for TCP amplification attack. These findings also prove that proper implementation of the virtual switch can help hypervisors prevent future attacks. In other words, to mitigate these attacks, the virtual switch should lower the retransmitting frequency and limit the abnormal infinite retransmissions.

SVTester was able to rediscover the possible vulnerabilities found in VMware Workstation Pro 12.0.0 [22]. Since this is an old version of VMware Workstation Pro, only companies that have not updated the hypervisor will be affected by the possible vulnerabilities found. However, the famous 2017 WannaCry ransomware attack that targeted Microsoft Windows XP and Windows Server 2003 proved that out-of-date software is still a continuing threat [14].

Furthermore, SVTester was able to identify a new possible vulnerability on a recent version of Oracle VirtualBox. This proves that recent versions of a hypervisor can have the same possible vulnerabilities as an older version of a different hypervisor.

Table 3 Testing tools comparison results.

Name	Possible Vulnerabilities		
	VMware NAT	Host PC	External TCP Host
OpenVAS	0	1	15
boofuzz	0	0	8
SVTester	2	0	0

Therefore, it is important to always properly implement the virtual switch and perform testings to prevent future attacks.

7.2 Comparison with Related Tools

In this section, we perform other tests on VMware Workstation 12.0.0 virtual network to verify whether existing testing tools can discover the possible vulnerabilities found by SVTester. First, we used OpenVAS version 21.04, a state-of-the-art vulnerability scanner [40]. OpenVAS has an easy-to-use interface and can perform in-depth network vulnerability scans by using daily updated feeds of network vulnerability tests with more than 80,000 plugins. We also performed other tests with a network protocol fuzzer called boofuzz (version 0.3), a fork of the famous Sulley fuzzing framework [41]. boofuzz also adopts the generation strategy but provides a much easier installation and better recording of test data than Sulley.

We implemented these tools on a virtual machine running Kali Linux 2021. In these testing experiments, we targeted the VMware NAT server, the Host PC running the hypervisor, and the external TCP Host we used to perform tests with SVTester (mentioned in Section 5.1). **Table 3** shows our comparison results. From our testing experiments, we can observe that OpenVAS and boofuzz are effective in finding vulnerabilities on a TCP Host. However, these tools could not explore the possible vulnerabilities that SVTester found in the VMware NAT because our tool is originally designed to exploit TCP retransmission. As a result, we can verify that SVTester can solve the problem that could not be discovered by existing methods.

7.3 Ethical Consideration

We have been careful to design experiments within legal and ethical boundaries. In particular, we only performed the attack experiments on our virtual network, which only had connections with our local subnet. Furthermore, we only sent a little amount of legal traffic to the entrusted TCP hosts used for research activities in our local subnet. Therefore, our experimental attack is harmless to the Internet.

We notified the abnormal behavior of VMware Workstation Pro 12.0.0's virtual switch to the vendor via email on 18 October 2018, However, we did not receive any response. Nonetheless, as mentioned in Chapter 7.1, the abnormal behavior of the virtual switch has been fixed in the latest version of VMware Workstation Pro. We also reported the abnormal behavior of Oracle VirtualBox 6.1.2's virtual switch and suggested some changes in the virtual switch configuration via their bug reporting site on 28 November 2020. However, as of 9 June 2021 which is over 5 months after our notice, we have not received any response. As mentioned in Chapter 6.2, the proposed attack based on the possible vulnerability showed only a slight effect on the targeted Oracle VirtualBox's system. Therefore, we believe that the impact of disclosing this possible vulnerability is minor.

7.4 Limitation

Since this is our first attempt to create a tool for evaluating the security of hypervisors against DoS attacks, SVTester still has some limitations. First, our tool has to send TCP packets from a VM, in other words, testers have to build a Linux OS VM inside the targeted system in order to execute SVTester. Second, SVTester requires installing a specific library (i.e., Scapy) for capturing TCP packets. Finally, only a few unusual thresholds are determined in SVTester, which results in a limitation in the number of possible vulnerabilities found from a virtual switch. We will revise this to further improve the capability of SVTester so that we can release a more complete version of it in the near future.

Our proposed attack schemes also have some limitations. The $S1$ attack scheme might be a minor threat to the hypervisor with the $V1$ possible vulnerability. Our experiment results show a small degradation in the performance of the other VM under the same targeted hypervisor. The $S2$ attack scheme might be a problem with hypervisors having the $V2$ possible vulnerability and a small cap of available ports in its PAT (for example VMware Workstation Pro 12.0.0). However, the $V2$ possible vulnerability does not affect the CPU, memory, and I/O performance of the other VM on the same hypervisor. Furthermore, the $S2$ schemes will find it difficult to occupy all of the available ports of the hypervisor with a large PAT capacity. However, we think it is difficult to fully judge the impact of the attack schemes with just one example and some performance benchmarks. Therefore, to further understand the impact of the proposed attack schemes, in the future we will improve the scale of the attacks and examine some different benchmarks such as web application requests serviced per second.

8. Conclusion and Future Work

In this paper, we described a new testing tool that can identify possible vulnerabilities exploited for certain types of DoS attack targeting virtual networks. We also perform testings on various hypervisors that implement NAT mode virtual switch for network connection. Testing results show that most of the recent hypervisors are unexploitable for amplification DoS attacks. We also found one novel possible vulnerability of Oracle VirtualBox latest version that can benefit DoS attackers. Comparison testing results show that SVTester is able to identify possible vulnerabilities that could not be found by existing testing tools. Our approach allows testers to build better testing tools to evaluate more possible vulnerabilities of virtual switches and hypervisors.

In the future, we plan to extend SVTester in a number of direction. First, we plan to implement more unusual thresholds so that SVTester could find more unexpected faults. For example, we expect that SVTester can determine whether the tested hypervisor can easily allow scanning for other VMs and spoofing IP address inside the virtual network. Second, we plan to use SVTester to test the security of more virtual infrastructures such as Docker or VPN. Finally, we plan to use SVTester to test the security of proxy servers since the implementation of a NAT mode virtual switch is similar to a physical proxy server.

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

References

[1] Henderson, A., Yin, H., Jin, G., Han, H. and Deng, H.: VDF: Targeted Evolutionary Fuzz Testing of Virtual Devices, *The 20th International Symposium on Research in Attacks, Intrusions and Defenses* (*RAID 2017*), Lecture Notes in Computer Science, Vol.10453, pp.3–25, Springer (2017).

[2] Oludele, A., Ogu, E.C., Shade, K. and Chinecherem, U.: On the Evolution of Virtualization and Cloud Computing: A Review, *Journal of Computer Sciences and Applications*, Vol.2, No.3, pp.40–43 (2015).

[3] Crane, C. and Nohe, P.: Re-Hash: The Largest DDoS Attacks in History (2020) (online), available from ⟨https://www.thesslstore.com/blog/largest-ddos-attack-in-history⟩

[4] Graziano, C.D.: *A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project*, Master of Science thesis, Iowa State University (2011) (online), available from ⟨http://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=3243&context=etd.⟩

[5] Rossow, C.: Amplification Hell: Revisiting Network Protocols for DDoS Abuse, *Symposium on Network and Distributed System Security* (*NDSS*) (2014) (online), available from ⟨https://christian-rossow.de/publications/amplification-ndss2014.pdf⟩

[6] Corero: Impact of DDoS on Enterprise Organizations (2018) (online), available from ⟨https://www.corero.com/blog/infographic-impact-of-ddos-on-enterprise-organizations/⟩

[7] Aschermann, C., Schumilo, S., Blazytko, T., Gawlik, R. and Holz, T.: REDQUEEN: Fuzzing with Input-to-State Correspondence, *26th Annual Network and Distributed System Security Symposium* (*NDSS 2019*) (2019) (online), available from ⟨https://www.ei.ruhr-uni-bochum.de/media/emma/veroeffentlichungen/2018/12/17/NDSS19-Redqueen.pdf⟩

[8] Commvault: CommVault Releases Results of Annual End-User Virtualization Survey Emphasizing the Need for Modern Approach to Protecting and Managing Virtual Server Environments (2011) (online), available from ⟨http://ir.commvault.com/news-releases/news-release-details/commvault-releases-results-annual-end-user-virtualization-survey?field_nir_news_date_value[min]=2018⟩

[9] 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⟩

[10] 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).

[11] 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).

[12] Banks, G., Cova, M., Felmetsger, V., Almeroth, K., Kemmerer, R. and Vigna, G.: SNOOZE: Toward a Stateful NetwOrk prOtocol fuzZEr, *9th International Conference on Information Security* (*ISC 2006*), Lecture Notes in Computer Science, Vol.4176, pp.343–358, Springer (2006).

[13] Yun, I., Lee, S., Xu, M., Jang, Y. and Kim, T.: QSYM: A Practical Concolic Execution Engine Tailored for Hybrid Fuzzing, *27th USENIX Security Symposium* (2018) (online), available from ⟨https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-yun.pdf⟩

[14] Kumar, M.S., Ben-Othman, J. and Srinivasagan, K.G.: An Investigation on Wannacry Ransomware and its Detection, *2018 IEEE Symposium on Computers and Communications* (*ISCC*), pp.1–6 (2018).

[15] Kührer, M., Hupperich, T., Rossow, C. and Holz, T.: Hell of a Handshake: Abusing TCP for Reflective Amplification DDoS Attacks, *8th USENIX Workshop on Offensive Technologies* (*WOOT'14*) (2014) (online), available from ⟨https://www.usenix.org/system/files/conference/woot14/woot14-kuhrer.pdf⟩

[16] 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⟩

[17] Abramov, R. and Herzberg, A.: TCP Ack Storm DoS Attack, *26th IFIP TC 11 International Information Security Conference* (*SEC 2011*) (2011) (online), available from ⟨http://dl.ifip.org/db/conf/sec/sec2011/AbramovH11.pdf⟩

[18] 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*) (2012).

[19] Schumilo, S., Aschermann, C., Abbasi, A., Wörner, S. and Holz, T.: HYPER-CUBE: High-Dimensional Hypervisor Fuzzing, *The Network and Distributed System Security Symposium* (*NDSS 2020*) (2020).

[20] Schumilo, S., Aschermann, C., Abbasi, A., Wörner, S. and Holz, T.: Nyx: Greybox Hypervisor Fuzzing using Fast Snapshots and Affine Types, *The 30th USENIX Security Symposium* (*USENIX Security'21*) (2021) (online), available from ⟨https://www.usenix.org/system/files/sec21summer_schumilo.pdf⟩

[21] Lee, S., Kim, J., Woo, S., Yoon, C., Scott-Hayward, S., Yegneswaran, V., Porras, P. and Shin, S.: A comprehensive security assessment framework for software-defined networks, *Computers & Security*, Vol.91 (2020).

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

[23] 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).

[24] 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).

[25] 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*) (2009) (online), available from ⟨https://hovav.net/ucsd/dist/cloudsec.pdf⟩

[26] Wang, Z., Zhang, Y. and Liu, Q.: RPFuzzer: A Framework for Discovering Router Protocols Vulnerabilities Based on Fuzzing, *KSII Trans. Internet and Information Systems*, Vol.7, No.8, pp.1989–2009 (2013).

[27] The Linux man-pages project: Linux Programmer's Manual - tcp(7): TCP protocol (online), available from ⟨https://linux.die.net/man/7/tcp⟩

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

[29] Amazon EC2, available from ⟨https://aws.amazon.com/ec2/?nc1=h_ls⟩

[30] iPerf3, available from ⟨https://iperf.fr/⟩

[31] Hyper-V Architecture, available from ⟨https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-architecture⟩

[32] Oracle VirtualBox, available from ⟨https://www.virtualbox.org/⟩

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

[34] PassMark PerformanceTest 10.1, available from ⟨https://www.passmark.com/products/performancetest/⟩

[35] QEMU 2.11.1, available from ⟨https://www.qemu.org/2018/02/14/qemu-2-11-1-and-spectre-update/⟩

[36] Scapy Project, available from ⟨https://scapy.net/⟩

[37] TCPView v3.05, available from ⟨https://docs.microsoft.com/en-us/sysinternals/downloads/tcpview⟩

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

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

[40] OpenVAS - Open Vulnerability Assessment Scanner, available from ⟨https://www.openvas.org/⟩

[41] boofuzz: Network Protocol Fuzzing for Humans, available from ⟨https://boofuzz.readthedocs.io/en/stable/⟩

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.