applied sciences

MDPI

Article

A VM-Based Detection Framework against Remote Code Execution Attacks for Closed Source Network Devices

Youngjoo Shin

School of Computer and Information Engineering, Kwangwoon University, 20 Kwangwoon-ro, Nowon-gu, Seoul 01897, Korea; yjshin@kw.ac.kr

Received: 27 February 2019; Accepted: 25 March 2019; Published: 28 March 2019

Abstract: Remote code execution attacks against network devices become major challenges in securing networking environments. In this paper, we propose a detection framework against remote code execution attacks for closed source network devices using virtualization technologies. Without disturbing a target device in any way, our solution deploys an emulated device as a virtual machine (VM) instance running the same firmware image as the target in a way that ingress packets are mirrored to the emulated device. By doing so, remote code execution attacks mounted by maliciously crafted packets will be captured in memory of the VM. This way, our solution enables successful detection of any kind of intrusions that leaves memory footprints.

Keywords: network intrusion detection; SDN/NFV; virtual machine; networked embedded systems; router and switches; virtualization technology

1. Introduction

The next-generation Internet drives a fundamental shift in the underlying networking architecture to enable dynamic deployment of applications in heterogeneous networks. Network virtualization, which is one of the key technologies, allows multiple logical networks to coexist on a shared hardware platform [1–4]. Virtualized network infrastructure introduces the idea of programmability in data plane of routers, which facilitates the dynamic deployment of applications and services [5,6]. In accordance with such technology trends, many networking hardware vendors provide virtual routers, such as Cisco CSR1000v Series and Huawei AR1000V routers, which can be deployed on a virtual machine (VM) on x86 commodity servers.

Implementing all layers of networking functions in virtual routers as software leads to the increase of vulnerabilities, which raises potential to compromise the network via a wide range of attack surfaces [7–9]. Sitting on layer 3, routers play a vital role in networking by connecting distinct networks and forwarding IP traffic across networks. Hence, once a router has been compromised, attackers may not only gain control over the entire network, but also obtain sensitive information by eavesdropping forwarded packets. As observed from the exposure by former NSA contractor Edward Snowden and the recently discovered Cisco IOS software attack dubbed SYNful Knock [10], the great potential of subverting network devices propels many criminals and state-sponsored hackers to pursue security vulnerabilities in routers for benefits.

Remote code execution attacks, caused by memory manipulation errors in software such as stack (or heap) overflow, comprise most infection vectors to compromise network devices [10,11]. Such kind of attacks affect control flow of the infected software, thus inherently leave footprints in the target's memory [12]. Host-based intrusion detection techniques can mitigate such memory corruption attacks by using the clues that influence the integrity of the memory.

To prevent network devices from being compromised by remote code execution attacks, intrusion detection system should be deployed to monitor the integrity of memory in network devices [9,13,14]. However, a key problem in network devices is that it is difficult or even infeasible to implement host-based detection techniques, which have been proven to be a successful solution in general purpose operating systems for a long time. That comes from the fact that a network device has a closed, special purpose operating system in which any third-party applications is not allowed to be installed and executed.

In this paper, we propose a novel intrusion detection technique for network devices. Our solution uses virtualization technology, and deploys a VM instance that emulates a target network device by running the same firmware image as the target. By configuring the network so that all traffic designated to the target are mirrored to the VM instance, all remote code executions triggered by malicious packets would be captured in memory of the virtual machine. This way, our solution provides a framework for successful detection of any intrusions leaving memory footprints.

The rest of this paper is organized as follows. Related work is summarized in Section 2, and design goals and challenges are presented in Section 3. Details of the system construction and the experiment as well as evaluation are given in Sections 4 and 5, respectively. Finally, we conclude the paper in Section 6.

2. Related Work

To monitor anomaly behaviors in networked embedded systems, Cui et al., proposed Symbiotes, a host-based intrusion detection system [15,16]. Their approach is to instrument CPU instructions causing a system exception at arbitrary locations in the target's firmware binary. Whenever these instructions are encountered during execution, context is switched to an integrity checking module, which also has been installed during the firmware modification. Unlike our proposed solution, Cui et al.,'s work forces a target device to run a modified firmware, which may affect its performance and reliability.

Another kind of works mainly address packet forwarding misbehavior detection for network devices [13,14,17–19]. These works are orthogonal to our approach, so can be combined in securing network devices.

3. Design Goals

To monitor any kind of remote code execution attacks against network devices and mitigate them without affecting performance of the devices, an intrusion detection system should ensure the following requirements:

- *Universality*: The system should succeed in detecting any kind of remote code execution attacks causing modifications in the target's memory.
- *Generality*: The system should support various platforms with diversity in vendors and operating systems.
- *Runtime support*: Any intrusion attempts to the target device should be detected immediately to mitigate the attacks in time.
- *Availability*: The system should work correctly while not affecting the internal states of the target device as well as its network configuration.

The fundamental challenge of an intrusion detection system for network devices is to provide an accurate detection of memory modification with low false positive (identifying remote code attacks that break the integrity of target's memory) [18]. Another important challenge of the system is to provide transparency over both the target device and the network. That is, the system must achieve its functional goals without affecting internal states of the target device as well as its network configuration (i.e., topology). To address the above challenges, we propose a VM-based intrusion detection system, which is described in detail in next section.

4. VM-Based Intrusion Detection Mechanism

In this section, we present the proposed VM-based intrusion detection system for network devices. We give a description of the system and its architecture, and present details on the system components.

4.1. System Architecture

The architecture of the proposed intrusion detection system is depicted in Figure 1. A target device, which would be a router in most deployment scenarios, has network connection with respect to its network configuration, and the connection is established via a L2 switch. The intrusion detection system is also connected with the switch via SPAN (Switched Port Analyzer). SPAN enables all forwarded packets through the target device to be duplicated and mirrored to the monitoring port which the monitoring system is attached to.

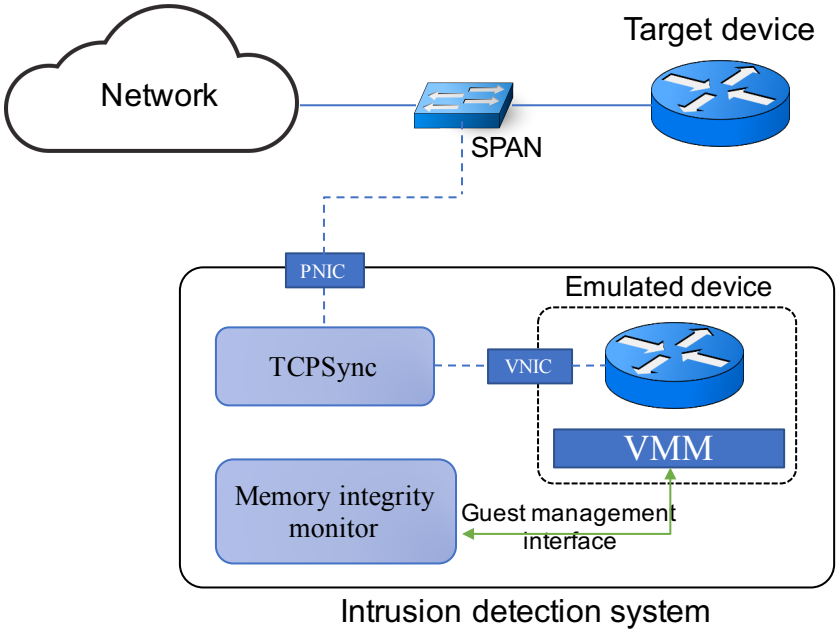


Figure 1. System architecture.

The proposed system, which is actually built on top of commodity x86 hosts, consists of the following components:

- *Emulated device*: It is a VM instance that emulates the target device. We require the VM to run the same binary image as the target's firmware. All mirrored packets from the target device, being passed through a physical network interface (PNIC) of the host system, TCPSync, and a virtualized network interface (VNIC) in order, are then injected into the emulated device. Due to the agreement of the firmware image and the traffic mirroring, internal states of the emulated device will conform to the target device.
- *TCPSync*: It is a software module, running outside the emulated device that forwards all received packets from a PNIC, transforming some packets if necessary, to a VNIC attached to the emulated device. The main work of this module is to conduct transparent conversion of certain packet flows through connection-oriented protocols such as TCP. Details on TCPSync will be presented in the next section.
- *Memory integrity monitor*: It is a software module that aims to monitor the internal state of the VM. It has an interaction with the emulated device through a guest management interface, which is provided by a virtual machine manager (VMM). By using an introspection library (e.g.,

Libvmi [20]) or a debugger such as GDB, it periodically obtains snapshots of the VM's memory and the corresponding hash value, from which it checks whether the integrity has been broken.

Figure 2 shows how a remote code execution attack against a target device can be detected by the proposed system. Suppose that the target device receives m ingress packets $p_1, p_2, ...p_i, ...p_m$ sequentially, among which a packet p_i contains maliciously crafted payload that will cause unauthorized modification of memory in the device. Those packets are mirrored to TCPSync, which in turn forwards the packets to the emulated device. Since a firmware image of the emulated device is identical to that of the target device, the crafted packet p_i will also cause memory corruption accordingly. Memory integrity monitor takes memory snapshots $S_0, ..., S_j, ...$ at every time epoch $T_0, ..., T_j, ...$ from the emulated device. It will find out that a snapshot S_j taken at time T_j after the crafted packet p_i has been injected has a variant in its hash value compared to previous snapshots, which indicates that memory corruption by the attack has been occurred.

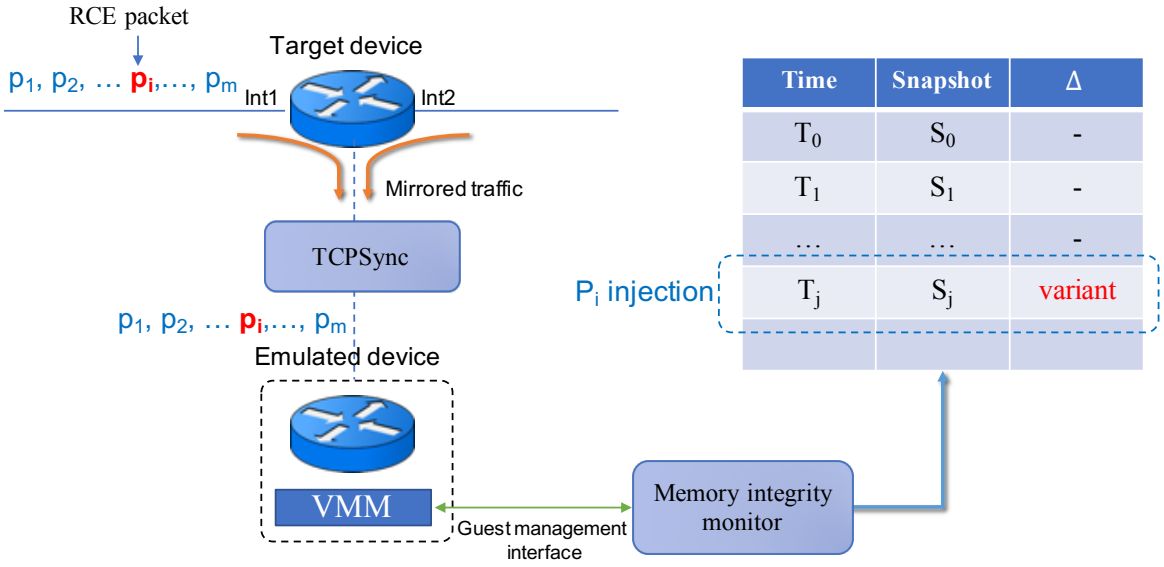


Figure 2. Example of a remote code attack detection.

4.2. System Components

We now take a look at how to construct each system component in this section.

4.2.1. Memory Integrity Monitor

Memory integrity monitor is a software module that keeps monitoring the internal state (i.e., the integrity of memory) of an emulated device during runtime. Specifically, it periodically obtains a copy of memory snapshot of the running VM, which contains an instance of an execution code and a runtime data at that time. Then it checks whether the integrity of the memory is maintained or not by a hash computation.

To obtain the snapshot, a memory integrity monitor makes use of a guest management interface which is offered by a VMM for virtual machine introspection. The guest management interface allows to monitor the low-level details of a running VM by viewing its entire memory content, trapping on hardware events, and accessing the CPU registers. This interface has a variety of implementations according to the types of VMM. For instance, in general purpose VMMs such as Xen, KVM, and QEMU, the interface is implemented as an introspection library (e.g., Libvmi [20]). In Dynamips, which is designed for emulating Cisco networking devices, a debugger such as GDB can be used as a guest management interface.

After obtaining a memory snapshot for the emulated device, the memory integrity monitor generates a hash value of the snapshot. The hash is computed by cryptographic hashing algorithms such as SHA-256. Then, it is compared to a previously computed hash for a former snapshot. The memory integrity monitor will be informed of an intrusion if there is a difference between two hashes, since it indicates the content of the captured memory has been altered.

4.2.2. TCPSync

Attack surfaces that cause memory corruption may originate from a diversity of network services ranging from L3 (e.g., ICMP) to L4 (e.g., FTP, HTTP, etc.), among which TCP-based protocols comprise a majority. Unlike other protocols, TCP is a connection-oriented, which means that each entity must maintain its own unique sequence number as an internal state for communication with each other. To the proposed system, this incurs a detection problem, since unless the same TCP state (i.e., sequence number) is used for both a target and an emulated device, the emulated device may fail to synchronize its memory with the target device.

For instance, certain Cisco IOS products are known to have memory corruption vulnerabilities in their FTP server [21]. Since an FTP service is based on TCP, an attacker first must establish a connection to the victim to exploit this vulnerability. Therefore, to follow each step needed for the attack to reappear in the emulated device, TCP sequence number must be synchronized with that of the target device.

TCPSync plays a role in addressing the problem of synchronizing TCP states. Residing on the path to an emulated device, this component performs transparent conversion of each mirrored TCP packet to adjust it to the consistent state of the emulated device.

Figure 3 illustrates how TCPSync works in the system. Suppose a TCP session between a target and a sender, where an initial sequence number of the target is set to be y. Because of all packets being mirrored, an emulated device also has sat up an initial sequence number z (not necessarily the same as y) during a three-way handshaking phase of this session. For every ingress TCP packet with an acknowledge number ACK = $y + \delta$ ($\delta \geq 0$), TCPSync converts it to ACK' = $z + \delta$ and then forwards the packet to the interface of the emulated device. For all this to work, TCPSync conducts packet analysis and obtains initial sequence numbers whenever a TCP connection is being established.

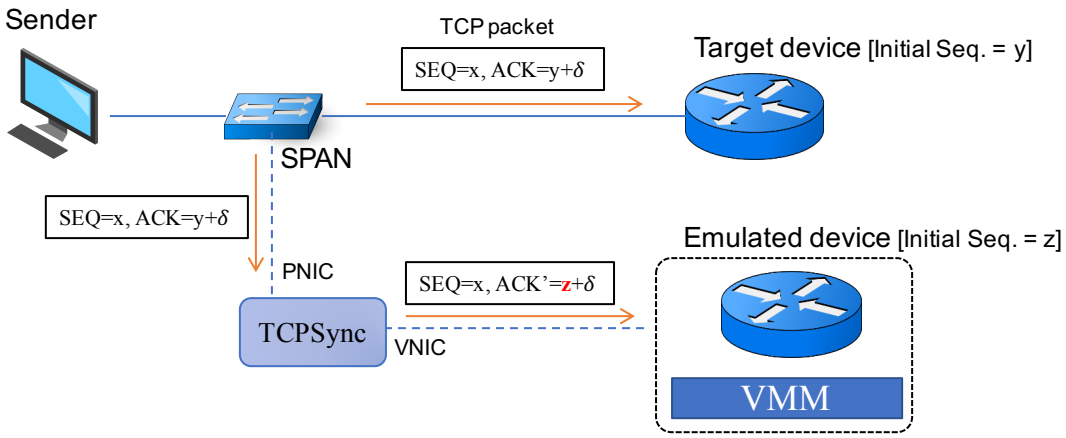


Figure 3. Transparent conversion of TCP packets by TCPSync (x, y, z are initial sequence numbers of a sender, target device, and emulated device, respectively).

5. Experiment and Evaluation

In this section, we present the experiment on the proposed VM-based intrusion detection system and evaluate it in terms of the aforementioned design goals.

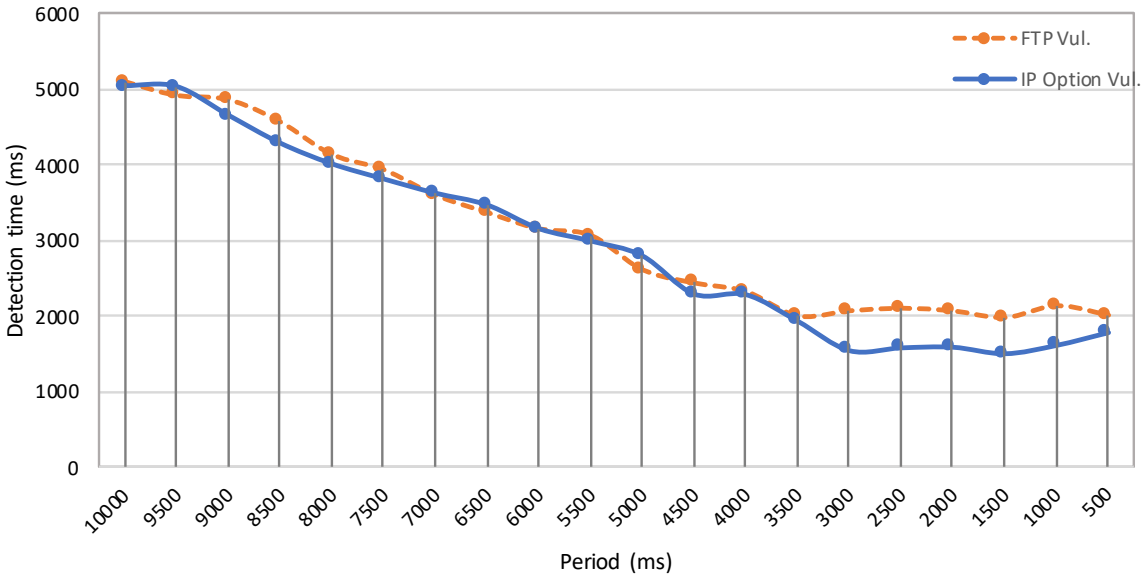
5.1. Experiment

We conducted some experiments to validate the effectiveness of the proposed intrusion detection system. In our experimental setup, a testbed consists of a Cisco 7200VXR router as a target network device connected to the Internet, and a x86 server that hosts the proposed detection system, both of which are connected through a L2 switch equipped with SPAN functionality. The testbed also has a laptop PC, connected to the router through the Internet, acting as a malicious user who delivers the remote code execution attack.

The Cisco router has an IOS firmware with a version of 12.3(6f), which has a couple of remote code execution vulnerabilities in the FTP server (CVE-2007-2586) [21] and an IP option handling routine (CVE-2007-0480) [22]. Both vulnerabilities allow unauthorized modification of the text segment of the firmware, resulting alteration of the control flow or injection of the shell code.

The host server is a machine with a 2.4 GHz Intel Core i5 CPU and 8GB RAM running a 64-bit Linux (Ubuntu 16.04 LTS). It runs the same firmware image as the target router via Dynamips, which is a MIPS emulator for Cisco routers. Dynamips provides a GDB interface for the purpose of debugging internals of the target device. Hence, the memory integrity monitor in our intrusion detection system can directly inspect the target's memory through the interface.

In our experiment, the laptop PC mounts an attack to the target router by sending crafted packets of the aforementioned vulnerabilities. Throughout the experiment, we measured the average detection time of the attack varying the interval at which the memory integrity monitor takes snapshots, as well as its performance overhead. Figure 4 shows the experimental result. The detection time steadily decreases as the interval of taking a snapshot is getting shorter. However, due to the inevitable delay for the memory dump, we need at least around 1500 ms to detect the attack (Figure 4a). Achieving faster detection comes at the cost of higher CPU use (Figure 4b). From the graph, we can infer that the period of 3000–3500 ms would give the optimal solution in terms of the detection time as well as the CPU consumption. It is worth noting that the attack exploiting the FTP server vulnerability needs more time in detection as well as more CPU than the other. This is because that the crafted packet for the FTP server should be constructed at the TCP layer, which requires the time-consuming operation of TCPSync.



(a) Detection time

Figure 4. *Cont.*

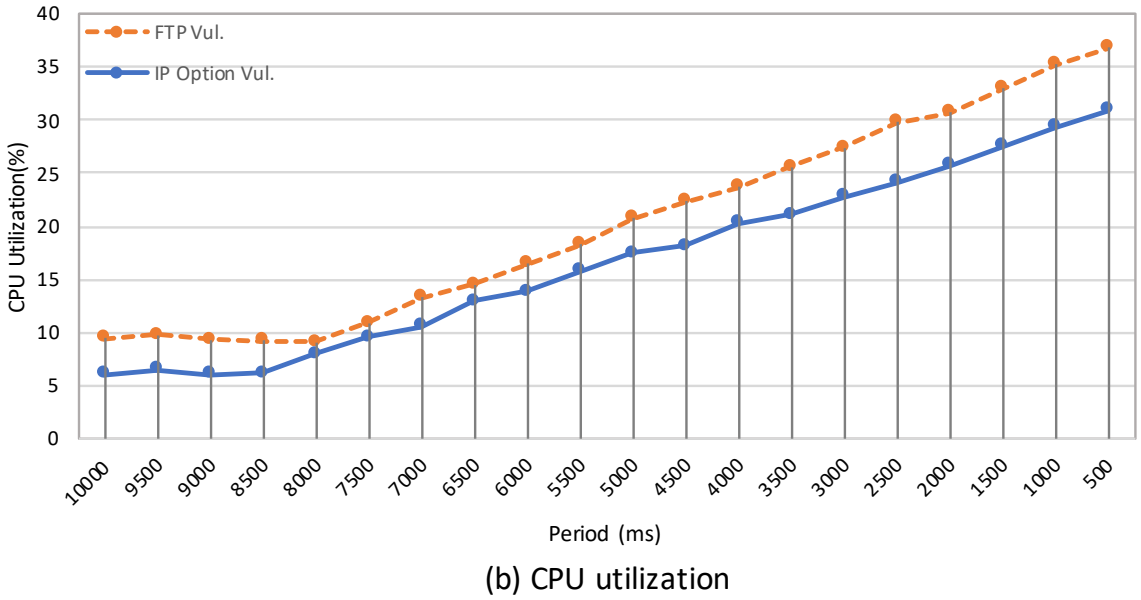


Figure 4. Experimental result.

5.2. Discussion on Detection Performance

In this paper, we consider attacks on network devices with more specific goals than any other kinds of attacks. That is, an attacker aims to implant a permanent backdoor into the network device by means of remote code execution attacks for the purpose of getting a persistent infiltration channel to the inside network. In fact, our consideration is based on analysis on recently unveiled attacks dubbed SYNful Knock [10], which targets Cisco devices. Such kind of attacks certainly affect control flow of the infected device, thus inherently leave footprints in the target's memory. Besides, the backdoor should remain implanted in the device for a long time according to the attacker's goal. These suggest that after the attack occurs, the internal state (i.e., integrity of the memory) of the infected device as well as its emulated VM remains inconsistent with the original benign state for a sufficient time to be detected by our proposed system. In this regard, we believe that it is enough with 2 s to detect such kind of attacks.

On the other hand, the attacker may try to disable the detection system after successfully taking control of the target device and before being alarmed by the system. However, the detection system is configured to investigate mirrored traffic without being connected to the network. It is infeasible for the attacker to turn off the alarm since there is no way to access the detection system over the network.

5.3. Evaluation

We insist that the proposed intrusion detection system achieves all design goals presented in Section 3.

Universality. Since internal states of a target and an emulated device are synchronized, all sort of remote code attacks that cause memory modification in a target device will be detected by memory integrity monitor.

Generality. Major network vendors such as Cisco and Huawei provide VM images of their virtualized routers in standardized format (e.g., OVA) to support various VMMs including VMware ESXi, Citrix Xen, or KVM. Since the proposed system has no dependency on any specific VMMs, generality can be achieved.

Runtime support. Memory integrity monitor periodically takes snapshots from an emulated device. As shown in the experiment in the previous section, the attack can be detected at least within around 2 s. The detection time is mainly determined from the delay of memory dump. Recent studies show that such delay is reasonable (e.g., dumping 1GB memory via GCORE takes about 1.5 s [12]), thus an event of remote code attack can be detected in time to sufficiently mitigate the attack.

Availability. Unlike previous works [15], the proposed system requires no modification to firmware of a target device. Besides, mirroring ingress traffic also does not affect the existing network topology. Therefore, the proposed system achieves availability.

6. Conclusions and Future Work

In this paper, we proposed a novel intrusion detection system for closed source networked embedded devices based on virtualized environments. Our solution deploys a VM instance that emulates the target device and makes all packets towards the target device are mirrored to the emulated device. By doing so, all remote code execution attacks delivered by maliciously crafted packets will be captured in memory of the VM without disturbing the target device. This way, our solution enables successful detection of any kind of intrusions that leaves memory footprints. As our further work, we will implement this system in real environments and evaluate its performance more rigorously to validate its effectiveness.

Funding: This research has been conducted by the Research Grant of Kwangwoon University in 2018 and was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No.2017R1C1B5015045).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Veeraraghavan, M.; Sato, T.; Buchanan, M.; Rahimi, R.; Okamoto, S.; Yamanaka, N. Network function virtualization: A survey. *IEICE Trans. Commun.* **2017**, *E100B*. [CrossRef]
2. Joshi, K.; Benson, T. Network Function Virtualization. *IEEE Internet Compu.* **2016**, *20*, 7–9.. [CrossRef]
3. Vilalta, R.; Muñoz, R.; Mayoral, A.; Casellas, R.; Martínez, R.; López, V.; López, D. Transport Network Function Virtualization. *J. Lightwave Technol.* **2015**, *33*, 1557–1564. [CrossRef]
4. Li, Y.; Chen, M. Software-defined network function virtualization: A survey. *IEEE Access* **2015**, *3*, 2542–2553. [CrossRef]
5. Martins, J.; Ahmed, M.; Raiciu, C.; Olteanu, V.; Honda, M.; Huici, R.; Felipe, B. ClickOS and the Art of Network Function Virtualization. In Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), Boston, MA, USA, 2–4 April 2014; pp. 459–473.
6. Egi, N.; Greenhalgh, A.; Handley, M.; Iannaccone, G.; Manesh, M.; Mathy, L.; Ratnasamy, S. Improved forwarding architecture and resource management for multi-core software routers. In Proceedings of the NPC 2009—6th International Conference on Network and Parallel Computing, Gold Coast, QLD, Australia, 19–21 October 2009; pp. 117–124. [CrossRef]
7. Scott-Hayward, S.; Natarajan, S.; Sezer, S. Survey of Security in Software Defined Networks. *Surv. Tutor.* **2016**, *18*, 623–654. [CrossRef]
8. Yin, Z.; Caesar, M.; Zhou, Y. Towards Understanding Bugs in Open Source Router Software. *ACM SIGCOMM Comput. Commun. Rev.* **2010**, *40*, 35–40. [CrossRef]
9. Prieto, L.P.; Rodríguez-Triana, M.J.; Kusmin, M.; Laanpere, M. Don't Trust Your Router: Detecting Compromised Routers. *CEUR Workshop Proc.* **2017**, *1828*, 53–59. [CrossRef]
10. Hau, B.; Lee, T.; Homan, J. SYNful Knock—A Cisco Router Implant. *Fire Eye Threat Res. Adv. Malware* **2015**. Available online: https://www.fireeye.com/blog/threat-research/2015/09/synful_knock_-_acis.html (accessed on 28 March 2019).
11. Li, F.; Zhang, L.; Chen, D. Vulnerability mining of Cisco router based on fuzzing. In Proceedings of the 2014 2nd International Conference on Systems and Informatics, ICSAI 2014, Shanghai, China, 15–17 November 2014; pp. 649–653. [CrossRef]

12. Sang-Hoon, C.; Ki-Woong, P. Mem-Shot: Design and Implementation of API-Trigger Driven Memory Dump System for Obfuscated Malware Analysis. *J. Korean Soc. Comput.* **2016**, *12*, 23–32.
13. Patil, R.; Modi, C. A Novel Approach to Detect Extraneous Network Traffic from the Compromised Router. In Proceedings of the 3rd International Conference on Recent Advances in Information Technology (RAIT), Dhanbad, India, 3–5 March 2016.
14. Mizrak, A.T.; Savage, S.; Marzullo, K. Detecting compromised routers via packet forwardåing behavior. *IEEE Netw.* **2008**, *22*, 34–39. [CrossRef]
15. Cui, A.; Stolfo, S.J. Defending Embedded Systems with Software Symbiotes. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 6961 LNCS, 2011; pp. 358–377. Available online: https://doi.org/10.1007/978-3-642-23644-0_19 (accessed on 27 March 2019).
16. Cui, A.; Kataria, J.; Stolfo, S.J. From prey to hunter. In Proceedings of the 27th Annual Computer Security Applications Conference on—ACSAC'11, Orlando, FL, USA, 5–9 December 2011; p. 393. [CrossRef]
17. Desai, V.; Natarajan, S.; Wolf, T. Packet forwarding misbehavior detection in next-generation networks. In Proceedings of the IEEE International Conference on Communications, Ottawa, ON, Canada, 10–15 June 2012; pp. 846–851. [CrossRef]
18. Le, F.; Lee, S.; Wong, T.; Kim, H.S.; Newcomb, D. Detecting network-wide and router-specific misconfigurations through data mining. *IEEE/ACM Trans. Netw.* **2009**, *17*, 66–79. [CrossRef]
19. Le, F.; Lee, S.; Wong, T.; Kim, H.; Newcomb, D. Minerals: Using Data Mining to Detect Router Misconfigurations. In Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data, Pisa, Italy, 11–15 September 2006; pp. 293–298. [CrossRef]
20. LibVMI-Project. Libvmi, a Virtual Machine Introspection Library. 2018. Available online: http://libvmi.com/ (accessed on 27 March 2019).
21. Cisco PSIRT. *Multiple Vulnerabilities in the IOS FTP Server*; Technical Report; 2007.
22. Cisco PSIRT. *Crafted IP Option Vulnerability in Cisco IOS*; Technical Report; 2007.