On Detection and Prevention of Zero-Day Attack Using Cuckoo Sandbox in Software-Defined Networks

Huthifh Al-Rushdan[1], Mohammad Shurman[2], and Sharhabeel Alnabelsi[3,4]
[1]Computer Engineering Depatmenr, Jordan University of Science and Technology, Jordan
[2]Network Engineering and Security Department, Jordan University of Science and Technology, Jordan
[3]Computer Engineering Department, Al-Balqa Applied University, Jordan
[4]Computer Engineering Department, AL Ain University, United Arab Emirates

Abstract: *Networks attacker may identify the network vulnerability within less than one day; this kind of attack is known as zero-day attack. This undiscovered vulnerability by vendors empowers the attacker to affect or damage the network operation, because vendors have less than one day to fix this new exposed vulnerability. The existing defense mechanisms against the zero-day attacks focus on the prevention effort, in which unknown or new vulnerabilities typically cannot be detected. To the best of our knowledge the protection mechanism against zero-day attack is not widely investigated for Software-Defined Networks (SDNs). Thus, in this work we are motivated to develop a new zero-day attack detection and prevention mechanism for SDNs by modifying Cuckoo sandbox tool. The mechanism is implemented and tested under UNIX system. The experiments results show that our proposed mechanism successfully stops the zero-day malwares by isolating the infected clients, in order to prevent the malwares from spreading to other clients. Moreover, results show the effectiveness of our mechanism in terms of detection accuracy and response time.*

Keywords: *Zero-day attack, Malwares, Controller, Intrusion Detection System, Cuckoo Sandbox, Software-Defined Networks.*

Received March 1, 2020; accepted June 9, 2020
https://doi.org/10.34028/iajit/17/4A/11

1. Introduction

Software-Defined Network (SDN) is a new approach that allows installing, controlling, managing, and modifying networks in a dynamic manner. SDN empowers a fast response to network requirements and can be managed using a centralized controller, such that switches and routers can be remotely reconfigured.

SDN architecture has three layers; first layer is the forwarding layer that consists of routers and switches. Second layer is the controller layer that consists of controllers. Third layer is the application layer that consists of application and services used to utilize SDN and generate traffic. SDN has two planes: a data plane and a control plane. The data plane operates under the open flow protocol plane that is responsible for forwarding packets, while the control plane decides the routing path for packets [9, 18].

When a packet arrives to the switch for the first time, a rule is inserted by the controller into the switch forwarding-table (a strategic control point in the SDN network that manages and controls the flows between network elements) [7], in order to deal with this packet, i.e., either forward it to a specific port or drop it. In fact, the switch sends all packets addressed to the same destination over the same route using the imposed rule.

The switch provides the controller with traffic information [6], where the communication between the controller and the switches is conducted using the OpenFlow protocol [21]. Thus, network administrator can centrally perform the necessary changes for the forwarding rules of switches (i.e., changing priorities or traffic blocking rules). Consequently, the popularity of SDN technology made it a target of security attacks.

In networks, there is still a risk of unknown attacks, known as the zero-day attacks [12, 15]. The term "zero-day" notion refers to the available time for vendors to fix the vulnerability that has been exposed [11]. When vendor fails to release a patch on time, the hacker can exploit the exposed vulnerability, and hence, the zero-day attack really occurs. The attacker executes a piece of code on the vulnerable system, in order to gain an illegal access. The term "vulnerability is exploited" occurs when an exploited code has successfully attacked the newly discovered vulnerability [3].

In general, in order to defend networks against unknown attacks, e.g., zero-day attack, an isolated testing environment, named a sandbox, is used to execute untested programs or files that may contain viruses or malicious codes, such that the real

environment is not infected [23], due to the fact that the sandbox is isolated from the real environment.

The sandbox contains a set of resources such as processors, memory, networks and applications that provided by a virtualization technique. In other words, the sandbox contains a number of Virtual Machines (VMs) with different Operating Systems (OSs). Each VM contains different programs, e.g., flash player, java. When a user downloads a file or visits a URL, the security system initially extracts the file or the URL, and then forwards it to the sandbox for execution on all VMs. Moreover, the sandbox sends some control instructions to the VMs, such as mouse movements or system time changes, because some malwares require special actions [23].

We are motivated to modify the existing sandbox, under UNIX OS, in order to integrate it with SDNs and protect their clients' PCs and controllers. This work is organized as follows: section 2 discusses SDNs security, section 3 presents the zero-day attack and section 4 presents Cuckoo sandbox analysis. Section 5 illustrates the proposed solution against zero-day attack. Section 6 demonstrates experiments results and discussion. Finally, section 7 presents the conclusions.

2. Security of Software-Defined Network

SDN security protocols are different from the standard networks, since its nature and characteristics are different. Therefore, SDN introduces new attacks to the controller platform and the connections between different planes. SDN treats control plane as a single entity, which indicates a single security implementation between the control plane and application plane, and between control plane and data plane.

Moreover, the implementation of distributed control is not visible to SDN architecture, since this may increase network exposure to attacks. On the other hand, the actual controller implementation is more complex and distributed, forcing stronger security requirements. These security requirements can be achieved by providing SDN controllers with a secure environment.

2.1. Preliminary

In network management, a real-time monitoring can be very useful, because it allows analysing and monitoring log entries, for forensic analysis, and intruders or attacks detection. It is possible to build SDN security techniques that combine stations and network devices security procedures, in order to detect and prevent attacks. One method of security procedures is isolating traffic between SDN users, and between users and control plan. This separation could be more effective and more dynamic than traditional networks, due to the processing and functional capability of data plane component.

The main security issues in SDN domain are the insiders and operator's errors that may compromise the overall system integrity. To address these issues, SDN architecture must contain a strong identity to secure all entities and their associated states [10], in addition to monitoring running processes.

2.2. Protection Methodology

SDNs do not change the associated protection and restoration protocol, such that their controllers are responsible for pre-computing resource recovery, provisioning recovery, and subscribing notification. Moreover, the SDN's controller may restore traffic by re-establishing the current route or selecting other routes to optimize utilized resources. These resources may be shared between more clients to satisfy their demand; therefore, resources must be fulfilled by a combination of the following procedures:

- Define a resource pool based on availability and recovery time, then serve clients accordingly.
- Protect the resources based on the most restricting requirement.
- Offer a default level of shared resources protection and provide clients with more restriction requirements.

3. Zero-Day Attack

The zero-day attack is a computer attack that exploits an exposed vulnerability, the vulnerability is a weakness in the software or in a security policy that allows the attacker to gain illegal access to the system that has not been known yet. Its aim is to get access or threat a running system [19, 22]. It is very difficult to defend against zero-day attack, since it is always detected after the system has been already compromised. The vulnerability, in zero-day attacks, has no known signature and no specific mechanism, which allows detecting and preventing it earlier [22]. Once the vulnerability has been announced to the public, system administrator can patch the system, and the antivirus companies can insert it in to the signature update [19].

Although, system patching, upgrading, antiviruses, and IDS can tackle many kinds of attack, the zero-day attack cannot be tackled, due to the lack of information about the attack's nature [5]. Discovering the zero-day vulnerability and figuring out how to stop it is a very difficult task. The zero-day vulnerability is considered as the most harmful threat for computer organizations, because their system and services are exposed to the public network and to the attacker before the patch becomes available. Researchers paid attention to zero-day attacks, in order to find solutions [1, 13, 20]. Generally, there are four kinds of traditional defense technology against attacks:

statistical-based, signature-based, behaviour-based, and hybrid-based [24].

4. Cuckoo Sandbox Analysis

Cuckoo sandbox has three VMs for testing which contains three versions of windows: Windows XP, Windows 7 and Windows 10. Once Cuckoo sandbox receives the files or the URLs for analysis, the process starts by restoring the current VM snapshot that contains a clean windows environment with several installed applications. Next, cuckoo sandbox will execute the file or open the requested URL in the browser, where the agent collects all changes in the VMs by profiling memory dump and registry information. After that, the agent transfers the collected changes to cuckoo sandbox for analysis by examining the memory dump, files created by the malware, and the registry information [23].

Once the analysis is completed, a report is generated for analysis result. The generated report has a score out of 10 representing the severity of attack for the file or the URL as follows:

- If (score < 2), this indicates the file or the URL is harmless.
- If ($2 \leq$ score ≤ 5), this indicates the file or the URL has high probability of being harmful.
- If (score > 5), then this indicates file or the URL is definitely harmful.

The score value will be sent to the controller to make an appropriate decision which is either isolating the client or blocking all its incoming traffic.

5. Proposed Solution against Zero-Day Attack

Security in SDN network is different from the traditional network security [2, 17] because the gateway of SDN is directly connected to the internal network, where all security devices are either installed in the application layer (controlled by the controller for traffic forwarding to the appropriate security device), or installed in the gateway layer (connected to the internal network). In this case, the controller has no control over the devices of the gateway [14]. Another main difference from traditional network is the controller controls every node in the network and can block the nodes' traffic or forward it to a specific path.

In this work, we implement a new system in SDN that protects two components:

1. The controller.
2. Client PCs against the zero-day attack. The proposed mechanism eliminates malwares' effect and protects the whole network from infection. We will use the mininet simulation tool [16], in order to implement SDN, forwarding switches, and the client PCs. Forwarding switches are connected to the controller using OpenFlow protocol [8], where the controller manages the traffic flow by forcing rules.

5.1. Client PCs Protection

In order to ensure client PCs protection, all traffic passes the OpenFlow switches goes through two stages: First, it is forwarded to the controller, on a specific network interface, where a customized python program extracts transferred files to the client or the requested URLs by the client. Once the files or the URLs have been extracted, the extraction program will submit them to the cuckoo sandbox for analysis, in order to detect malwares, if exist.

5.2. SDN Controller Protection

The protection of SDN controller from zero-day attack is different from the client protection. Cuckoo sandbox only tests the malware under windows environment, while the controller is usually based on UNIX environment that is not supported by Cuckoo. Therefore, we are strongly motivated to build our UNIX-based sandbox.

Our developed sandbox controller consists of an agent installed on a VM, as an SDN controller, and an application runs on a machine, which is hosting the controller, where the communication between the sandbox and the controller will be carried out through a dedicated Ethernet channel.

In our study, the agent will monitor three main parts that affect the status of the controller:

- Added or removed features to/from the controller.
- The status of the service port in the controller.
- The status of specific service in the controller's OS.

If any feature is changed, thus a new attack has occurred. The flowchart that demonstrates the steps of our developed controller sandbox is illustrated in Figure 1.

The steps in the flowchart, as shown in Figure 1, are explained as follows:

- *Step* 1: Sandbox with Controller in VM: The sandbox is installed on Ubuntu OS and runs a server software that manages all operations and controls the VM's controller, the server contains VMWare Workstation 12 Pro virtualization software. The controller's software is Open Daylight (ODL) Hydrogen version.
- *Step* 2: The operational controller with the installed agent: Hydrogen ODL is installed with a number of features, based on the networks requirements that required for running the controller. The installed features should run in the active state, in order to ensure that the controller is in the operational state.

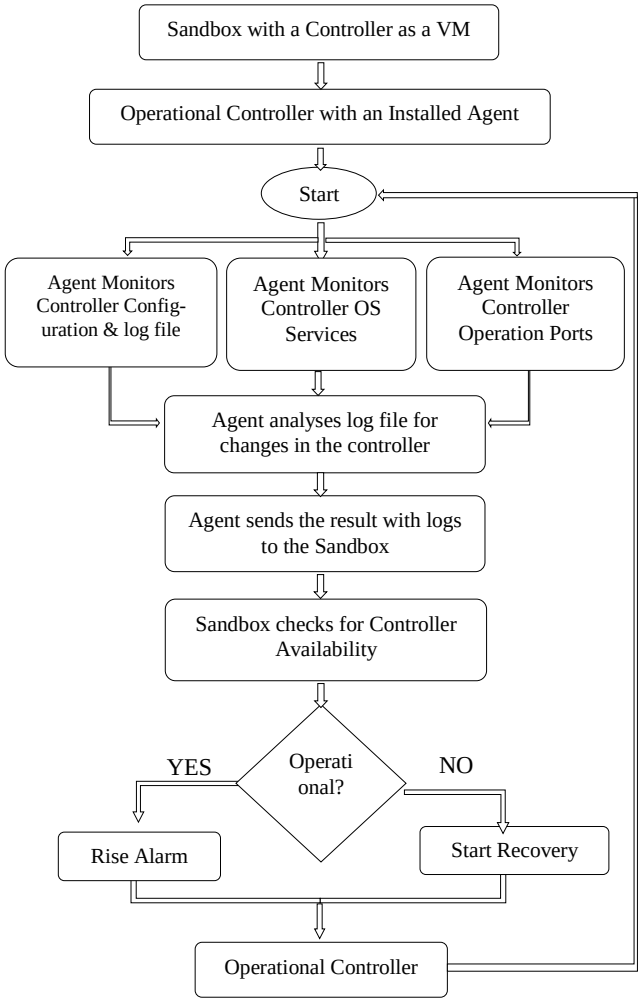

Figure 1. The steps for our developed controller sandbox.

The ports associated with the installed features should be opened and in the listening state, such that they are closed when their features are removed. Apparently, a specific software is needed, and therefore, is installed in order to monitor the controller from attacks. That software is called the agent.

- *Step* 3:

a) Agent monitors controller configuration (log file): When a feature is installed, it is parameters and configurations are stored in the configuration files of the controller. Therefore, installing or removing any feature illegally results in compromising some controller's functions. For example, if the authentication feature is removed, thereby, any connection at the OpenFlow switch can add or remove flows at the controller. If a specific feature is installed without a proper configuration, this may lead to illegal controller's operation. Consequently, the agent always monitors adding and removing any feature as in the controller's configuration files.

b) Agent monitors controller's services: The controller is installed in a Linux based OS, and it use several Linux services such as NTP, Java, etc. If any of these services failed or changed its status, the controller will stop working. Thus, the agent monitors all Linux services that are crucial to the controller operation, such that if any service is stopped the agent will try to restart this service. However, if this service is failed to restart, the agent triggers an alarm to the controller and transmits the log file for analysis.

c) Agent monitors controller operation ports: The controller uses specific ports for operation, e.g.; OpenFlow switches communicates with the controller over port 6653. If an attack destroys the corresponding features of this port, the controller will no longer be listening to the port requests. Therefore, the agent must monitor the operational ports, such that if the controller failed to respond three consecutive times to the controller, the agent starts analysing the log files and triggers an alarm to cuckoo sandbox.

- *Step* 4: Agent analyzes log file for changes in the controller:
  If any alarm was triggered in step 3, the agent starts analysing the log file in order to find out the problem, and if there is any illegal access. The log files that will be analysed are as follows:
  - Opendaylight.log: this file has all feature installation and removal in the Open Day light controller, such that if any feature was initialized or destroyed it is profiled.
  - tomcat0.log: this file profiles the status of features whether they are changed from running to stopping or from stopping to running state. The agent analyses this file to find the status of the corrupted feature in the controller.
  - audit.log: this file profiles the status of accessing the controller, a success or failure. The agent finds whether there are any illegal access attempts.
  - web_access_log.log: this file profiles any access attempt, using either web browser or REST API access. The agent finds any link between this access and feature corruption or any illegal flow insertion.
- *Step* 5: The agent sends the log file to cuckoo sandbox with the results:
  The agent sends the analysis result with all log files to the sandbox for more analysis and starts the recovery procedure, if needed.
- *Step* 6: Sandbox checks for controller availability:
  The sandbox will check the availability of the controller using these steps:
  1. Checking all controller's operational ports that are open and in listening state.
  2. Checking agent log for services status.
  3. Installing a static flow in the controller.
  4. Sniffing on the controller traffic to ensure that the controller installed the flow in the OpenFlow switch.
  5. Analysing the agent logs to find out what is really happening.

6. Raising an alarm to the administrator with the results.

- *Step* 7: Rise an alarm to further investigation:
  If the controller passed all the tests in step 6, raise an alarm to the administrator with all logs for deeper investigation.
- *Step* 8: Start Recovery Procedure: If the controller failed with any of the performed tests in step 6, it will conduct the following procedures:

  1. Pull all controller log file and save it for future analysis.
  2. Clone the controller VM for more investigation.
  3. Restore the controller VM to the previous snapshot.
  4. Raise an alarm to the administrator with the attack.

- *Step* 9: Operational Controller: the controller is back to its operational status monitored by the agent.

For the aforementioned steps, each step has a linear-time complexity, O(N), where N is the number of its internal steps. Therefore, the overall time complexity for all steps is also linear.

6. Experimental Results and Discussion

6.1. Platforms

Simulation experiments are conducted using these tools:

- Client Machines: Intel® Core™2 i5-3230M CPU, 8 GB DDR3 RAM, which is used to download the malware and act as infected client.
- Controller Sandbox: Intel® Core™2 i7-4770M CPU, 24 GB DDR4 RAM, which has the application that monitors the SDN controller and the virtualization software that hosts the VM controller.
- Controller: Intel® Core™2 i7-4770M CPU, 4 GB DDR4 RAM, a virtual machine that installed with the controller software and the agent which monitors the SDN controller.
- Cuckoo Host: Intel® Core™2 i7-4770M CPU, 12 GB DDR4 RAM, equipped with the cuckoo software.
- Cuckoo VM: Intel® Core™2 i7-4770M CPU, 2 GB DDR4 RAM, a virtual machine that used by cuckoo sandbox to test the malware.
- Switch: OpenFlow switch v1.0.

Figure 2 shows the testing environment for our proposed solution, in which SDN client is protected from the zero-day attack. Figure 3 shows the environment used to test the proposed solution that protects the SDN's controller from the zero-day attack.

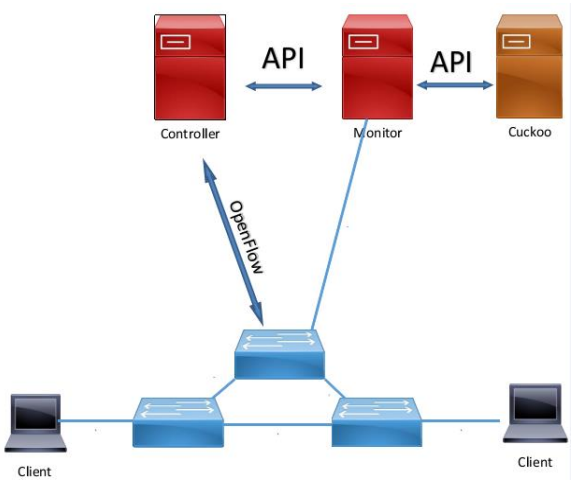

Figure 2. Testing environment for SDN client.

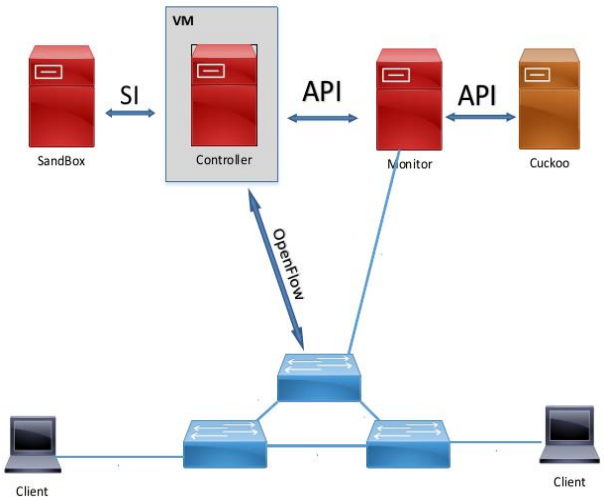

Figure 3. Testing environment for SDN controller.

6.2. Results

Python 3.5 programming language is used under Linux, in order to implement all required functions of our proposed mechanism.

For clients' PCs protection, we use Snort tool which is a free open source network Intrusion Detection System (IDS) and Intrusion Prevention System (IPS). This tool contains a feature that allows us to extract the files from live network traffic stream and the corresponding client IP address. Once the file is extracted, our developed python program sends the file to the cuckoo sandbox for analysis and wait the result. After the sandbox finishes the analysis, these results are sent to the python program, such that if the results indicate an attack has occurred, the python program blocks the client PC using the controller API.

For SDN controller protection, the agent monitors the controller features, listed in subsection 5.2, in order to detect any change that caused by the zero-day attack. The functionality of the controller depends on the installed features, where initially the controller begins with zero features installed. Later, for example, during the operation, if the controller needs to handle open flow switches, the odl-l2switch-switch feature is

installed. Another example, if GUI interface is needed, the odl-dlux-core feature is enabled.

Different python libraries are used, such as socket for interfacing, system-specific parameters and functions, shutil for file operations, threading parallelism, watchdog observers, and others.

We introduce three performance metrics to assess the proposed mechanism effectiveness as follows:

1. *Execution time*: is the total execution time starting from the moment that the malware is download to the moment when the controller finishes blocking the infected client.
2. *Analysis time*: this represents the cuckoo sandbox analysis time which includes: the summation processes, malware execution inside the VM, collecting data, analysing data, and result analysis and summation.
3. *Processing (or blocking) time*: is required time for processing sandbox resultant data, which includes the malware file reconstruction and controller blocking process. Thus, the processing time = total execution time – cuckoo sandbox analysis time.

The preliminary results of this work have been presented in [4]. For cuckoo sandbox, Table 1 shows the number of successfully identified malwares, with different sizes, with respect to number of tested malwares. Results show that cuckoo sandbox has successfully identified 353 malwares of 361. Thus, the success percentage is 97.78%, which can be further improved by customizing cuckoo sandbox configuration.

Table 2 shows the test result of the controller for sandbox, a total of 50 tests are conducted for each service type. Our proposed scheme succeeded to recover the controller 192 times of 200 (96%), and failed 8 times out of 200 (4%).

Figure 4 shows the malware analysis time with respect to malware's size. Results show the execution time is proportional to the malware size, moreover, the analysis time is about 132 s and 152 s when the malware size is 2 KB and 1400 KB, respectively. Apparently, even when malware size increases dramatically from 2 KB to 1400 KB, the analysis time increases only 15.1%. Clearly, this demonstrates the effectiveness of our proposed technique.

The system blocking time includes file reassemble, analysis result submission, network communication, and controller blocking process. Figure 5 shows the system blocking time (or processing time) for infected clients with respect to different malware's sizes. As illustrated, clearly this time is negligible, because for different malware sizes it is below 0.01 s. As a result, our proposed system reacts to a zero-day malware and blocks it very fast regardless of the malware size when the analysis results received from cuckoo sandbox. However, the system's recovery time varies with respect to the damage caused by the malware, because the modified files must be restored.

Table 1. Number of tested and successfully identified malwares.

Experiment number	Malware size (KB)	Number of tested malwares	number of successfully identified malwares
1	2	20	19
2	30	18	18
3	50	22	21
4	70	22	20
5	80	19	19
6	90	20	20
7	100	17	16
8	150	16	16
9	200	20	20
10	300	19	19
11	400	18	17
12	500	17	16
13	600	19	19
14	700	20	20
15	800	20	20
16	900	21	21
17	1000	13	13
18	1200	21	21
19	1400	19	18
Total		**361**	**353**

Table 2. Number of succeeded and failed tests for different services.

Test type	Number of tests	Succeeded	Failed
Remove Feature	50	48	2
Remove Port Service	50	50	0
Change Controller Configuration	50	46	4
Stop Linux Service	50	48	2
Summation	**200**	**192**	**8**

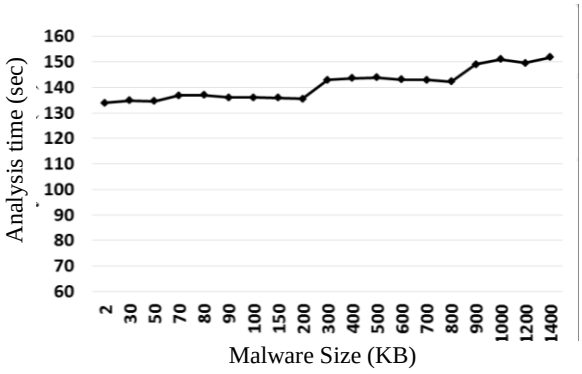

Figure 4. Analysis time for different malware sizes.

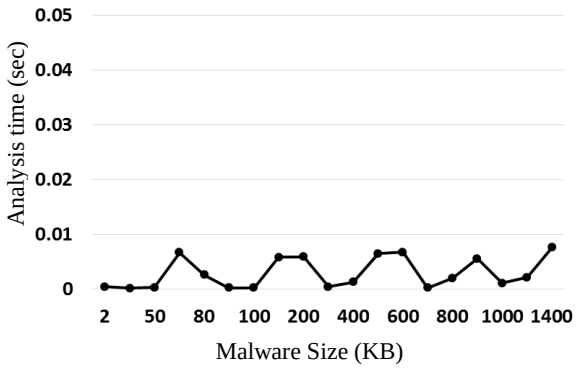

Figure 5. System blocking time with respect to malware size.

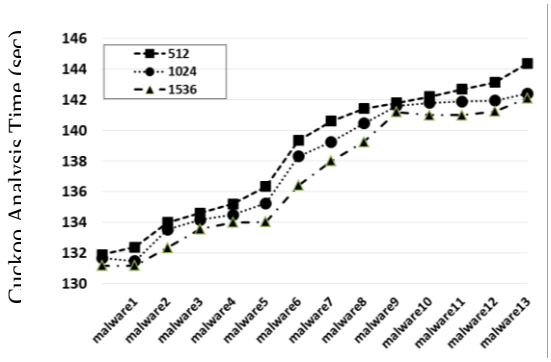

Figure 6. Analysis time for different malware's and RAMs sizes.

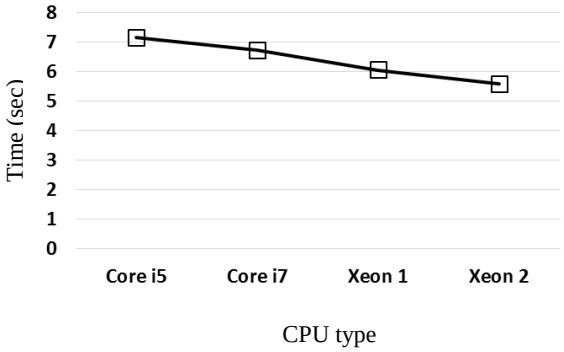

Figure 7. Controller recovery time for different CPU's types.

Intuitively, reacting to the malware and blocking it is very fast, simply by informing the firewall to block it and dispose its corresponding code from the memory.

Figure 6 shows the cuckoo sandbox analysis time for the first 13 malwares, which presented in Table 1 using different RAM sizes in MB (512, 1024 and 1536) of the tested VM's. Apparently, malware analysis time is faster with bigger RAM size. That is, when using a VM with higher capabilities, e.g.; RAM size, for hosting the cuckoo sandbox. Consequently, the infected client is blocked before the malware finishes execution and attacks other clients.

Figure 7 shows the recovery time for the controller according to the processor speed, results shows that the controller recovery time is faster with higher host CPU speed.

Our proposed approach computation complexity as evaluated by the simulation is very low and almost negligible. That is due to the fact that the time complexity of the proposed system includes: trace file processing (N_1 bytes), memory dump processing (N_2 bytes), files created by the malware extraction or URL extraction process (N_3 bytes), and finally, reporting the generated file (constant time of value C), where N_1, N_2, and N_3 are integer numbers.

As presented in subsection 5.2, the internal steps within each process has a linear-time complexity. Consequently, the time complexity of the system is the summation of time-complexities for all mentioned independent stages as: $O(N_1)+O(N_2)+O(N_3)+O(C)$. That is, the time complexity is linear, O(M), where M is equal to $N_1 + N_2 + N_3 + C$.

In our simulation model, analysis time is found between 130-150 seconds, this variation in analysis time is due to malware and RAM sizes. In our simulation model, we considered one zero-day malware. In the future, we will test the model for multiple zero-day malwares and investigate the system performance.

7. Conclusions

The zero-day attack has a severe effect on Software-Defined Networks (SDNs), especially, it is unpredictable. This kind of attacks exposes undiscovered networks vulnerability, in order to get illegal access to the network and cause harmful effect. Moreover, software developers have a zero-day, in order to resolve this attack and protect the network.

The proposed mechanism that based on cuckoo sandbox identifies and prevents malwares within a zero-day time, in order to protect two components: First, the clients' PCs that are protected by our customized-developed python code that resides in the controller. Second, attacks on SDN controller which is prevented using our proposed and the developed UNIX-based Cuckoo sandbox. In this controller, traffic is monitored by the imposed detection rules.

Experimental results show our proposed mechanism is effective in detecting different malwares attacks, and has a high success probability in identifying different malwares. Moreover, the results demonstrate that the blocking time is negligible when our proposed technique is employed, also the analysis time increases slightly when the malware size increases. However, when utilizing VMs with high capabilities, the analysis times decreases.

References

[1] Afek Y., Bremler-Barr A., and Feibish S., "Zero-Day Signature Extraction for High-Volume Attacks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 2, pp. 691-706, 2019.

[2] Alauthman M., Aslam N., Al-Kasassbeh M., Khan S., Al-Qerem A., and Choo K., "An Efficient Reinforcement Learning-Based Botnet Detection Approach," *Journal of Network and Computer Applications*, vol. 150, pp. 102479, 2020.

[3] Almukaynizi M., Nunes E., Dharaiya K., Senguttuvan M., Shakarian J., and Shakarian P., "Proactive Identification of Exploits in the Wild Through Vulnerability Mentions Online," *in Proceedings of IEEE International Conference on Cyber Conflict*, Washington, pp. 82-88, 2017.

[4] Al-Rushdan H., Shurman M., Alnabelsi S., and Althebyan Q., "Zero-Day Attack Detection and Prevention in Software-Defined Networks," *in Proceedings of the International Arab*

Conference on Information Technology, Alain, pp. 278-282, 2019.

[5] Bilge L. and Dumitras T., "Before We Knew It an Empirical Study of Zero-Day Attacks in The Real World," *in Proceedings of ACM Conference on Computer and Communications Security*, Raleigh North Carolina, pp. 833-844, 2012.

[6] Braun W. and Menth M., "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," *Journal of Future Internet*, vol. 6, no. 2, pp. 302-336, 2014.

[7] Doria A., Salim J., Haas R., Khosravi H., Wang W., Dong L., Gopal R., and Halpern J., *Forwarding and Control Element Separation (ForCES) Protocol Specification*, RFC 5810, pp. 1-124, 2010.

[8] Goto Y., Ng B., Seah W., and Takahashi Y., "Queueing Analysis of Software Defined Network with Realistic Openflow-Based Switch Model," *Computer Networks*, vol. 164, pp. 301-306, 2019.

[9] Haleplidis E., Denazis S., Koufopavlou O., Salim J., and Halpern J., "Software-Defined Networking: Experimenting with the Control to Forwarding Plane Interface," *in Proceedings of the European Workshop on Software Defined Networks*, Darmstadt, pp. 91-96, 2012.

[10] Karakus M. and Durresi A., "A Survey: Control Plane Scalability Issues and Approaches in Software-Defined Networking (Sdn)," *Computer Networks*, vol. 112, pp. 279-293, 2017.

[11] Kaur R. Singh M., "A Survey on Zero-Day Polymorphic Worm Detection Techniques," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 3, pp. 1520-1549, 2014.

[12] Keramati M., "An Attack Graph Based Procedure for Risk Estimation of Zero-Day Attacks," *in Proceedings of The 8th International Symposium on Telecommunications*, Tehran, pp. 723-728, 2016.

[13] Kim J., Bu S., and Cho S., "Zero-Day Malware Detection Using Transferred Generative Adversarial Networks Based on Deep Autoencoders," *Information Sciences*, vol. 460, pp. 83-102, 2018.

[14] Kreutz D., Ramos F., and Verissimo P., "Towards Secure and Dependable Software-Defined Networks," *in Proceedings of the 2nd ACM SIGCOMM workshop on Hot Topics In Software Defined Networking*, China, pp. 55-60, 2013.

[15] Meneely A. and Lucidi S., "Vulnerability of the Day: Concrete Demonstrations for Software Engineering Undergraduates," *in Proceedings of the 35th International Conference on Software Engineering*, San Francisco, pp. 1154-1157, 2013.

[16] Rashma B. and Poornima G., "Performance Evaluation of Multi Controller Software Defined Network Architecture on Mininet," *in Proceedings of the International Conference on Remote Engineering and Virtual Instrumentation*, Switzerland, pp. 442-455, 2019.

[17] Sachdeva M., Singh G., Kumar K., and Singh K., "DDoS Incidents and their Impact: A Review," *The International Arab Journal of Information Technol*ogy, vol. 7, no. 1, pp. 14-20, 2010.

[18] Shin M., Nam K., and Kim H., "Software-Defined Networking (SDN): A Reference Architecture and Open APIs," *in Proceedings of the International Conference on ICT Convergence*, Jeju Island, pp. 360-361, 2012.

[19] Singh U., Joshi C., and Singh S., "Zero-day Attacks Defense Technique for Protecting System Against Unknown Vulnerabilities," *International Journal of Scientific Research, Computer Science and Engineering*, vol. 5, no. 1, pp. 13-18, 2017.

[20] Singh U., and Joshi C., and Kanellopoulos D., "A Framework for Zero-Day Vulnerabilities Detection and Prioritization," *Journal of Information Security and Applications*, vol. 46, pp. 164-172, 2019.

[21] Sood M., "Software Defined Network-Architectures," *in Proceedings of International Conference on Parallel Distributed and Grid Computing*, Solan, pp. 451-456, 2014.

[22] SDN Architecture, Open Networking Foundation, Technical Report, 2016.

[23] Vasilescu M., Gheorghe L., and Tapus N., "Practical Malware Analysis Based on Sandboxing," *in Proceedings of RoEduNet Conference 13th Edition: Networking in Education and Research Joint Event RENAM 8th Conference*, Chisinau, pp. 1-6, 2014.

[24] Wang L., Zhang M., Jajodia S., Singhal A., and Albanese M., "Modeling Network Diversity for Evaluating The Robustness of Networks Against Zero-Day Attacks," *in Proceedings of the 19th European Symposium on Research in Computer Security*, Wroclaw, pp. 494-511, 2014.

Huthifh Al-Rushdan received his B.Sc. degree in Computer Engineering, Jordan University of Science and Technology, Jordan, 2007. He received his M.Sc. in Computer Engineering, Jordan University of Science and Technology, 2018. Currenly, he is head of datacenters in Jordan Army. His research interests are in SDN, compuer security, datacenters, computer networks and virtualization.

Mohammad Shurman received his B.Sc. degree in Electrical and Computer Engineering from Jordan University of Science and Technology, Irbid, Jordan, 2000. Also, he received his M.Sc. and Ph.D. degrees in Computer Engineering-Wireless Networks from University of Alabama-Huntsville (UAH) in 2003 and 2006, respectively. Presently, he is with the Network Engineering and Security Department, Jordan University of Science and Technology, Irbid, Jordan. His research interests include wireless Ad-hoc networks, security and key management of wireless networks, wireless sensor networks, network coding, wireless communication and mobile networks, software defined networks (SDN), cognitive radio, WiMAX, 4G and 5G technologies and Blockchains.

Sharhabeel Alnabelsi is an associate professor at Computer Engineering Dept. at Al-Balqa Applied University, Amman, Jordan. Also, he is an associate professor in Computer Engineering Dept. at Al Ain University, UAE. He received his Ph.D. in Computer Engineering from Iowa State University, USA, 2012. Also, he received his M.Sc. in Computer Engineering from The University of Alabama in Huntsville, USA, 2007. His research interests are cognitive radio networks, wireless sensors networks, network resources optimization, and cloud computing. He is a member of honorary societies including Eta Kappa Nu and Phi Kappa Phi.