SD-MTD: Software-Defined Moving-Target Defense for Cloud-System Obfuscation

Ki-Wan Kang[1], Jung Taek Seo[2], Sung Hoon Baek[3], Chul Woo Kim[4], and Ki-Woong Park[1,*]
[1] Dept. of Information Security, and Convergence Engineering for Intelligent Drone, Sejong University, Seoul 05006, Korea
[e-mail: woongbak@sejong.ac.kr]
[2] Department of Computer Engineering, Gachon University
Gyeonggi-do 13120, Korea
[e-mail: seojt@gachon.ac.kr]
[3]Department of Computer System Engineering, Jungwon University
Chungcheongbuk-do 28024, Korea
[e-mail: shbaek@jwu.ac.kr]
[4]LG CNS,
Magok JoongAng St 8, Korea
[e-mail: enjoy0124@gmail.com]
[*]Corresponding author: Ki-Woong Park

Received February 4, 2022; revised March 3, 2022; accepted March 11, 2022; published March 31, 2022

Abstract

In recent years, container techniques have been broadly applied to cloud computing systems to maximize their efficiency, flexibility, and economic feasibility. Concurrently, studies have also been conducted to ensure the security of cloud computing. Among these studies, moving-target defense techniques using the high agility and flexibility of cloud-computing systems are gaining attention. Moving-target defense (MTD) is a technique that prevents various security threats in advance by proactively changing the main attributes of the protected target to confuse the attacker. However, an analysis of existing MTD techniques revealed that, although they are capable of deceiving attackers, MTD techniques have practical limitations when applied to an actual cloud-computing system. These limitations include resource wastage, management complexity caused by additional function implementation and system introduction, and a potential increase in attack complexity. Accordingly, this paper proposes a software-defined MTD system that can flexibly apply and manage existing and future MTD techniques. The proposed software-defined MTD system is designed to correctly define a valid mutation range and cycle for each moving-target technique and monitor system-resource status in a software-defined manner. Consequently, the proposed method can flexibly reflect the requirements of each MTD technique without any additional hardware by using a software-defined approach. Moreover, the increased attack complexity can be resolved by applying multiple MTD techniques.

Keywords: Cloud Computing System, Container Orchestration, Moving-Target Defense, System Obfuscation.

This work was supported by the Defense Acquisition Program Administration and Agency for Defense Development under the contract UD210029TD.

http://doi.org/10.3837/tiis.2022.03.017 **ISSN : 1976-7277**

1. Introduction

Container techniques are applied to existing cloud computing systems to maximize their efficiency, flexibility, and economic feasibility [1–3]. Concurrently, studies on ensuring the security of cloud computing systems are gaining attention [4, 5]. Current cloud computing systems generally maintain initially defined system attributes in a static manner [6, 7]. This provides attackers with sufficient time and information to analyze the vulnerabilities of target systems [8, 9]. To address these vulnerabilities, moving-target defense (MTD) techniques, which increase the cost and complexity of attacks by periodically or aperiodically varying the attack surface of the protected target, have emerged [10, 11]. Cloud computing systems, which offer a high agility and flexibility, enable the implementation of various MTD techniques. Specifically, network-based MTD techniques are effective as they make the attacker's initial reconnaissance step difficult and reverse the information asymmetry between attackers and defenders [12–15].

However, although each MTD technique is capable of deceiving attackers, they have limitations when applied to actual cloud-computing systems. These include wasted resources, management complexity caused by the additional function implementation and system introduction, and an increase in attack complexity that may occur because of the application of a single MTD technique. Accordingly, this study proposes a software-defined MTD system, SD-MTD, that can flexibly apply and manage existing and future MTD techniques without infringing on economic feasibility [16, 17]. Prior to the design and implementation of the proposed SD-MTD system, this study derives its requirements. First, the implemented MTD techniques should be centrally managed and applied. Second, it should be possible to verify a valid range of mutations using the implemented MTD techniques. Third, it should be possible to apply multiple MTD techniques to specific services to address a potential increase in the attack complexity of a single MTD technique. Based on these derived requirements, an SD-MTD system composed of an SD-MTD dashboard, an SD-MTD orchestrator, an SD-MTD agent, and connector modules was designed and implemented.

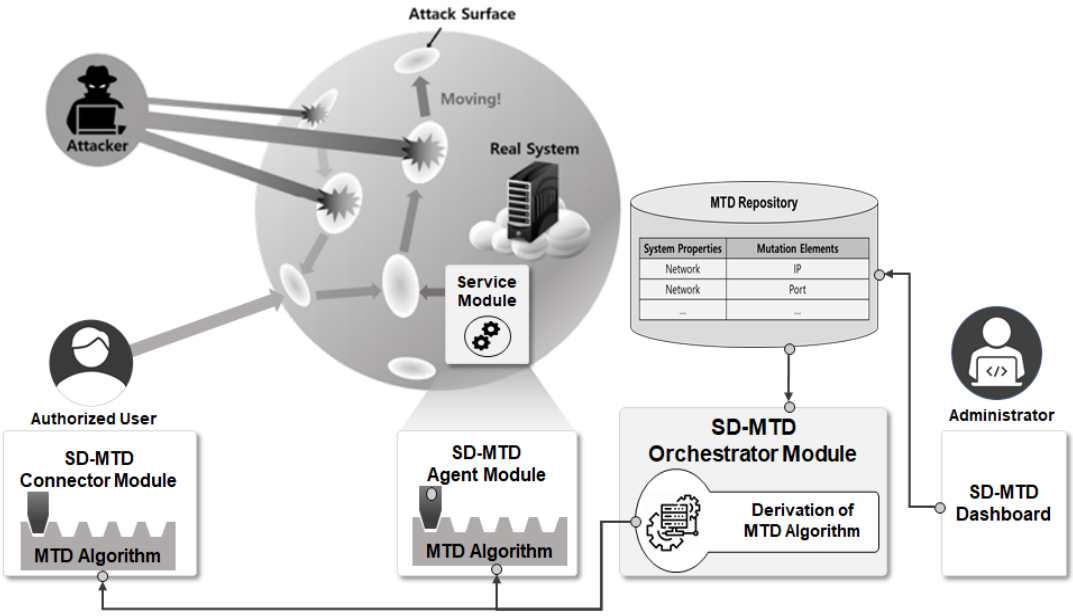

Fig. 1. Overall system architecture and core components of SD-MTD system

Fig. 1 shows the overall architecture of the SD-MTD system that can flexibly apply and manage existing and future MTD techniques. The proposed system is designed to correctly define a valid mutation range and cycle for each moving-target technique in a software-defined manner (by the SD-MTD orchestrator module in **Fig. 1**) and monitor the system-resource status (by the SD-MTD dashboard in Fig. 1). Consequently, the proposed method can flexibly reflect the requirements of each MTD technique, without requiring additional hardware, by using a software-defined approach [18, 20].

The contributions of this study can be summarized as follows. First, user-defined customization is secured by enabling the cloud computing system administrator to flexibly manage the valid mutation range and cycle for the mutation elements of the implemented MTD. Second, an orchestrated control is ensured by enabling the administrator to flexibly manage and apply the implemented MTD to the protected service. Third, scalability is ensured by allowing multiple MTD techniques [21–26] to be applied flexibly according to the security level of the service provided through the cloud computing system.

The remainder of this paper is organized as follows. Section 2 analyzes and discusses the limitations of existing studies on MTD techniques. Section 3 derives the SD-MTD system requirements and describes its design and implementation. Section 4 presents the verification results of proposed SD-MTD system's accuracy. Finally, Section 5 concludes the study and presents the future research directions.

2. Related Works

According to the mutation strategy, MTD techniques can be classified as shuffling-based, diversity-based, or redundancy-based approaches. Shuffling periodically or aperiodically rearranges or randomly mutates the main attributes of a system. Diversity periodically or aperiodically mutates the system configuration, for example, the software operating environment. Redundancy restores normal services by duplicating the main system attributes in advance and replacing them with duplicated system attributes in a post-attack scenario. Shuffling- and diversity-based MTD techniques primarily prevent an attack even if the target information is exposed. In contrast, redundancy-based MTD techniques employ post-measurement methods.

2.1 MTD based on system-attribute shuffling

In this section, we discuss the main results and characteristics of the system attribute-shuffling-based MTD.

- ***Random port and address hopping (RPAH):*** Luo et al. [27] predefined a mutation cycle and proposed a technique for changing the IP address and port number according to the defined mutation cycle. They used a pseudo-random number generator to generate virtual addresses for mutation. To practically apply this technique, the function must be implemented in the target gateway to be protected. However, collisions may occur because the randomly generated IP address and port number do not validate the current in-use IP address and port number.

- ***Ghost-MTD:*** Park et al. [28] proposed ghost-MTD (gMTD), which mutates a protocol using a one-time bit sequence (OTBS), which is shared securely in advance. When OTBS is employed for mutation, only authorized service users who have information

on the protocol-mutation pattern can communicate with the service module. Further, gMTD can deceive an attacker by redirecting it to a decoy-hole module when the access is abnormal. To apply this method to an actual operating system, it is necessary to implement an additional technique for safely exchanging OTBSs.

2.2 MTD based on system-attribute diversity

In this section, we discuss the main results and characteristics of the system attribute diversity-based MTD.

- ***Moving attack surfaces (MAS)-based MTD:*** Huang et al. [29] proposed a technique to mutate the software stack based on the mutation cycle of events. They utilized several web services, web servers, operating systems, and virtual technologies for mutation. Only authorized service users who have information on the elements configured to provide web services can use normal services. To apply this method to an actual environment, it is necessary to construct virtual servers implemented with several web servers, operating systems, and virtual technologies for the service to be protected.

- ***Programming language diversity (PLD)-based MTD:*** Taguinod et al. [30] proposed a technique that mutates the web programming language for providing web services and SQL language for managing stored data. The mutation is done by implementing a translator, which mutates the SQL language by converting Python code into PHP code and moving the data stored in MySQL to PostGreSQL. In an actual environment, the limitations of the translator's code conversion and the time required to transfer a large amount of data can be a hindrance. Furthermore, important functions in the program operation may be omitted during the mutation.

2.3 Limitations of MTD techniques

An analysis of existing MTD techniques revealed that, although each technique is capable of deceiving attackers, their capabilities are limited when applied to actual cloud-computing systems. First, they degrade the resource efficiency and management convenience of a cloud-computing system because of their additional function implementation and system introduction. Second, an MTD should be applied without disturbing the current operating service. Third, it should be possible to address an increased attack complexity that may be encountered when a single MTD is applied.

3. Design and Implementation of SD-MTD

3.1 Derivation of SD-MTD requirements

Cloud computing, which integrates container techniques to maximize efficiency, flexibility, and economic feasibility, has attracted a significant attention. Concurrently, emerging MTD techniques can reverse the information asymmetry of attackers and defenders in cloud-computing systems with a high agility and flexibility. Accordingly, a system is required to apply and manage the existing and future MTD techniques. Therefore, this study proposes a software-defined MTD (SD-MTD) system to address an increased attack complexity by applying a new technology based on an advanced MTD to an existing system. Prior to

designing and implementing the proposed SD-MTD, the following requirements should be derived:

- ***User-defined customization:*** The MTD attribute to be applied for system defense and mutation range and mutation cycle of each MTD attributes should be flexibly defined and applied by the system administrator in a timely manner. This will help the system administrator respond flexibly to a rapidly changing security environment.

- ***Orchestrated control:*** A system enhanced with MTD technology should be adaptable not only to conventional MTD technologies but also to emerging MTD technologies. To address this issue, it is necessary to develop a technology to flexibly accommodate each MTD technology and orchestrate them centrally using software-defined technology.

- ***Scalability:*** Conventional MTD techniques are limited in the face of an increased attack complexity because their system resources are not scalable. This limitation can be overcome by dynamically applying an MTD technology suitable for the protected service according to the required security level and situation.

The design philosophy of SD-MTD is based on the above three requirements, and the design and implementation for realizing each requirement will be described in the following section.

3.2 SD-MTD system design for cloud-system obfuscation

This section describes an SD-MTD system design that satisfies the requirements established in the previous section. To ensure user-defined customization, orchestrated control, and scalability, the SD-MTD system is composed of an SD-MTD dashboard, SD-MTD orchestrator, SD-MTD agent, and SD-MTD connector modules. **Fig. 2** shows the overall architecture and conceptual operation flow of the system. Each component and operation flow are described as follows:

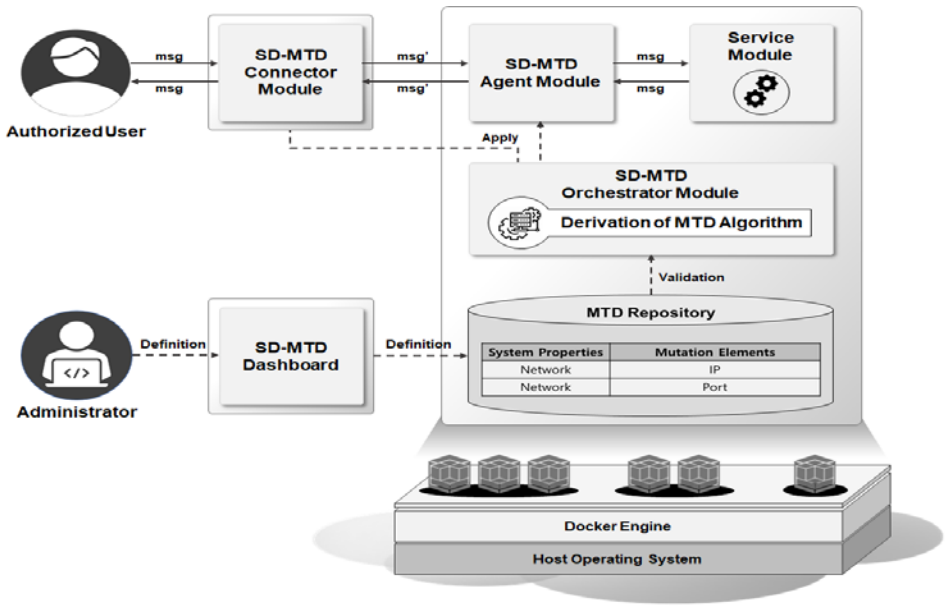

Fig. 2. Main components of SD-MTD and overall operation flow to realize MTD

- ***Ensuring user-defined MTD customization:*** The administrator can check the implemented MTDs in advance, and then select the desired MTD technique to be applied through the SD-MTD dashboard, as shown in **Fig. 2**. Subsequently, the administrator defines a valid mutation range and cycle for the selected MTD. The administrator-defined MTD is then delivered to the SD-MTD orchestrator module, which generates a mutation list after verifying whether the administrator-defined valid mutation range is correct. Thus, the SD-MTD system flexibly reflects the requirements of each MTD technique by enabling the execution of administrator-defined MTDs.

- ***Ensuring orchestrated control:*** When an MTD technology and its mutation rules are confirmed by the administrator, its deployment will be formally described in the MTD repository. Subsequently, the SD-MTD orchestration module generates scripts to perform the MTD's execution and verification routines. The generated scripts are deployed and delivered to the SD-MTD agent and SD-MTD connector modules. The above end-to-end routines are continuously managed and validated by the SD-MTD orchestration module. Therefore, this study ensures an orchestrated control by enabling the application of an MTD selected by the administrator through the SD-MTD system.

- ***Ensuring scalable MTD:*** The SD-MTD agent and SD-MTD connector modules operate the MTD routines delivered from the SD-MTD orchestration module. However, depending on the security situation or available system resources, the MTD technologies to be applied may be dynamically changed. Consequently, the SD-MTD system is designed to ensure scalability by applying container and orchestration technologies that can dynamically change and respond to the system on demand.

The design of the SD-MTD system can be summarized as follows. First, user-defined MTD customization is ensured by enabling the system administrator to flexibly define a valid mutation range and cycle for the mutation elements of the MTD to be applied. Second, orchestrated control is ensured by enabling the administrator to flexibly manage and apply the implemented MTD to the service to be protected. Third, scalability is ensured by flexibly applying multiple MTD techniques according to the security level of the service provided through the cloud computing system.

3.3 SD-MTD implementation for cloud-system obfuscation

This section describes the implementation of the SD-MTD system for existing and future MTD techniques. The structure of the implemented SD-MTD system is shown in **Fig. 2**. It is composed of an SD-MTD dashboard, SD-MTD orchestrator, SD-MTD agent, SD-MTD connector, and service modules. The SD-MTD dashboard is provided for the SD-MTD administrator, the SD-MTD connector module is provided for the authorized user, and the SD-MTD orchestrator, SD-MTD agent, and service modules are provided for the cloud-computing system.

- ***SD-MTD dashboard:*** The *SD-MTD* dashboard is provided to the administrator. In the dashboard, we can define the valid mutation range and cycle of a previously

implemented MTD. Furthermore, the MTD can be implemented according to the defined valid mutation range and cycle, and then applied to the service to be protected. It is necessary to verify the validity of the MTD implemented by the administrator before its application to the service to be protected. To achieve this, the MTD implemented through the *SD-MTD* dashboard is delivered to the *SD-MTD* orchestrator module, which verifies its validity.

- ***SD-MTD orchestrator module:*** The *SD-MTD* orchestrator module is provided to a container-based cloud computing system. It verifies that the MTD is implemented through the *SD-MTD* dashboard. The resources of the container-based cloud-computing system to which the *SD-MTD* is applied are monitored to verify the valid mutation range of the MTD implemented from the *SD-MTD* dashboard. For example, when applying an MTD to IP shuffling, the MTD is implemented by considering whether the IP address being used is within the valid mutation range inputted from the *SD-MTD* dashboard. Based on this, the MTD implemented in the *SD-MTD* dashboard is verified, and a mutation list is generated. Subsequently, the list is transmitted to the *SD-MTD* agent and *SD-MTD* connector modules to apply it to the communication.

- ***SD-MTD agent and SD-MTD connector modules:*** The *SD-MTD* agent and *SD-MTD* connector modules are delivered to the container-based cloud-computing system and the authorized service user, respectively, and the MTD implemented from the *SD-MTD* dashboard is applied to the communication. Communication is performed according to the mutation list, which is generated when the valid mutation range defined by the administrator is determined to be correct through the *SD-MTD* orchestrator module, and according to the mutation cycle defined through the *SD-MTD* dashboard. The *SD-MTD* agent and *SD-MTD* connector modules obfuscate the communication between the cloud-computing system and the authorized service user according to the MTD defined by the administrator. Further, the *SD-MTD* agent and *SD-MTD* connector modules verify the obfuscated communication; if the communication is determined to be correct, it is converted into a normal communication result and delivered to the cloud system and service user.

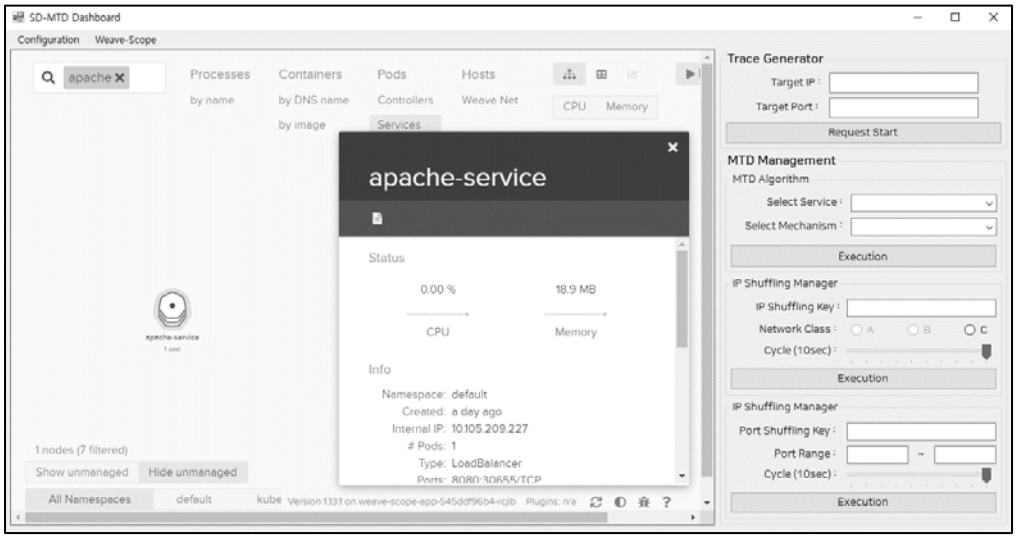

Fig. 3. Dashboard interacting SD-MTD testbed for performance evaluation

- ***Service module:*** This is a service provided to authorized service users. The service execution is based on a container and it be operated in various forms, such as web services. Unauthorized service users can perform malicious activities by exploiting the vulnerabilities in the cloud-computing system and service modules. Thus, the MTD is employed to ensure the security of service modules. In this study, the Apache web service, which is operated in actual cloud-computing systems, is implemented to verify the accuracy of the proposed *SD-MTD* system.

4 Testbed Construction and Evaluation

4.1 Testbed for SD-MTD evaluation

IP shuffling and port shuffling, which are network-based MTDs, were implemented to verify the accuracy of the SD-MTD system. The implemented IP shuffling and port shuffling each generate a mutation list according to the key values and valid mutation range that are input and defined through the SD-MTD dashboard and then obfuscate the communication according to the defined mutation cycle. The dashboard provides an interface in the Windows environment to continuously request the Apache service and check whether the applied MTD is operating. The administrator's convenience was enhanced by implementing a GUI environment to connect with the implemented MTD, rather than accessing the virtual machine with the MTD implementation. The SD-MTD is composed of request generation, MTD management, and visualization functions, and the implemented SD-MTD dashboard is shown in **Fig. 3**. First, a function checks the execution status of the network-based MTD, which is implemented through continuous requests. Second, the valid mutation range and cycle are configured for the implemented IP shuffling and port shuffling. Then, the MTD is applied according to the configured valid mutation range and cycle. Third, the CPU, memory consumption, IP address, and port number of the Apache service are verified, and the resource status of the experimental environment is monitored.

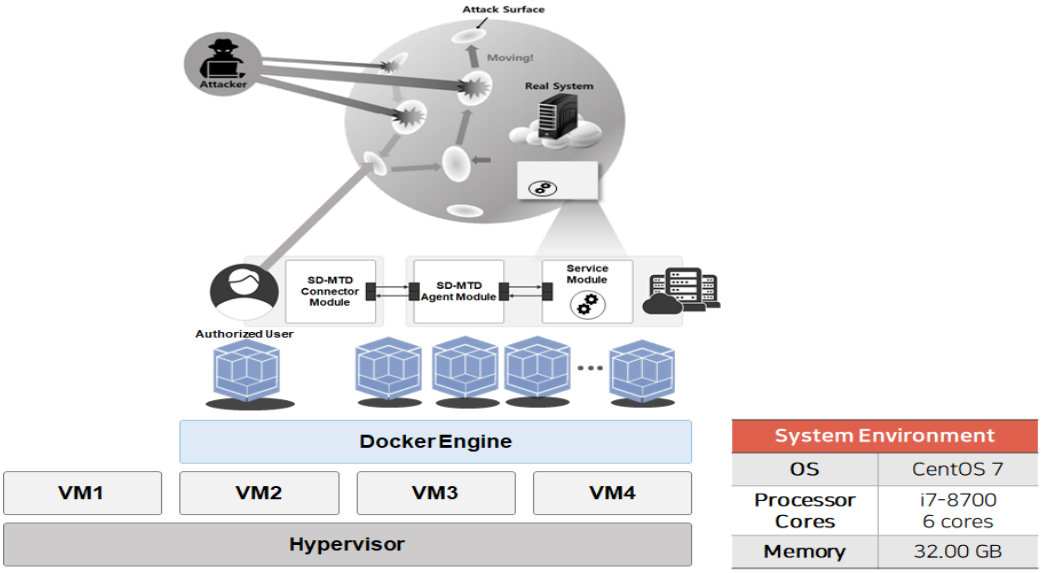

Fig. 4. Experiment environment for accuracy verification of the SD-MTD

4.2 Functional verification of the SD-MTD implementation

We measured the behavior accuracy of the SD-MTD system. We conducted an experiment on a server equipped with an Intel(R) Core™ i7-8700 and 32 GB of RAM. For this experiment, we created user and server system virtual machines (VMs) with 2 vCPUs, 2 GB memory, and Centos 7 OS, as shown in **Fig. 4**.

4.2.1. Functional verification of the SD-MTD user-defined customization and centralized control

This experiment defined the valid mutation range and cycle of the MTD through the SD-MTD dashboard and evaluated whether it was executed according to the defined values. In **Fig. 5(a)**, the valid mutation range and mutation cycle are defined for IP shuffling. The network C class was configured with a 10 s mutation cycle for the experiment. In **Fig. 5(b)**, the valid mutation range and cycle are defined for port shuffling. The port was configured with a 20 s mutation cycle in the port number range of 60,000–65,535 for the experiment. In a correct SD-MTD implementation, a valid mutation range and cycle should be defined, and the MTD should be executed according to the defined values. Accuracy verification of the user-defined customization and centralized control of the SD-MTD system revealed that the MTD was executed correctly according to the defined mutation range and mutation cycle.

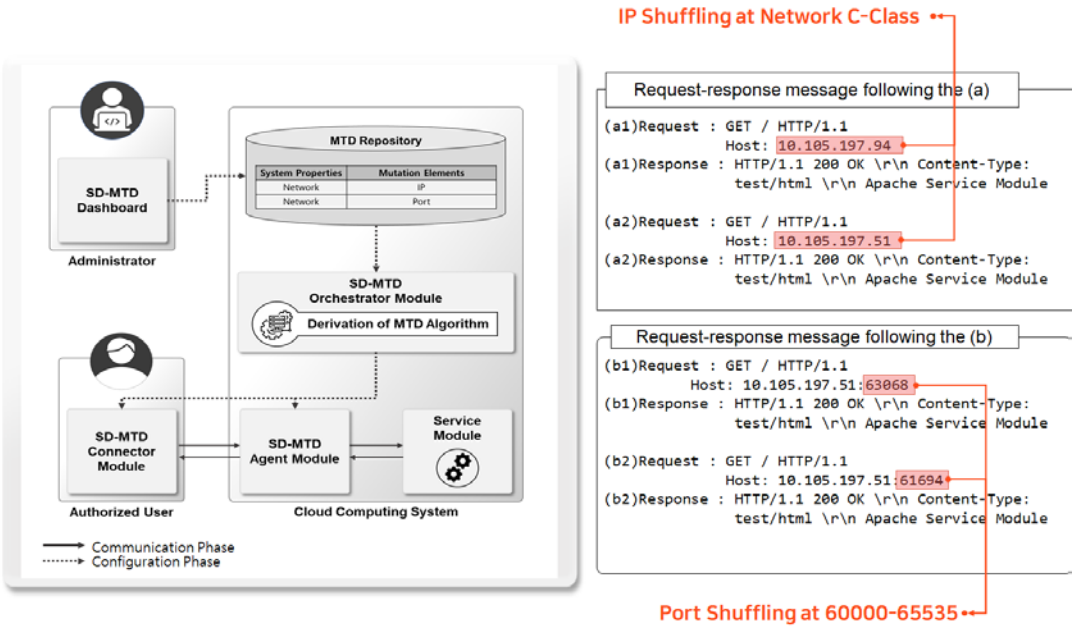

Fig. 5. Request-response messages for (a) IP shuffling and (b) port shuffling

4.2.2 Functional verification of the SD-MTD scalability

In this experiment, multiple MTD techniques were applied to the Apache service through the SD-MTD dashboard, and each MTD technique was checked for its operating status. **Fig. 6** illustrates the application of the implemented IP shuffling and port shuffling to the Apache service. To conduct the experiment, an arbitrary network class and mutation cycle were defined and the setup was verified to evaluate whether the MTD techniques were executed according to the defined values. In a correct SD-MTD implementation, multiple MTD

techniques are simultaneously applied, and each MTD technique should be executed. Accuracy verification of SD-MTD scalability revealed that multiple MTD techniques were applied to the Apache service and operated accordingly.

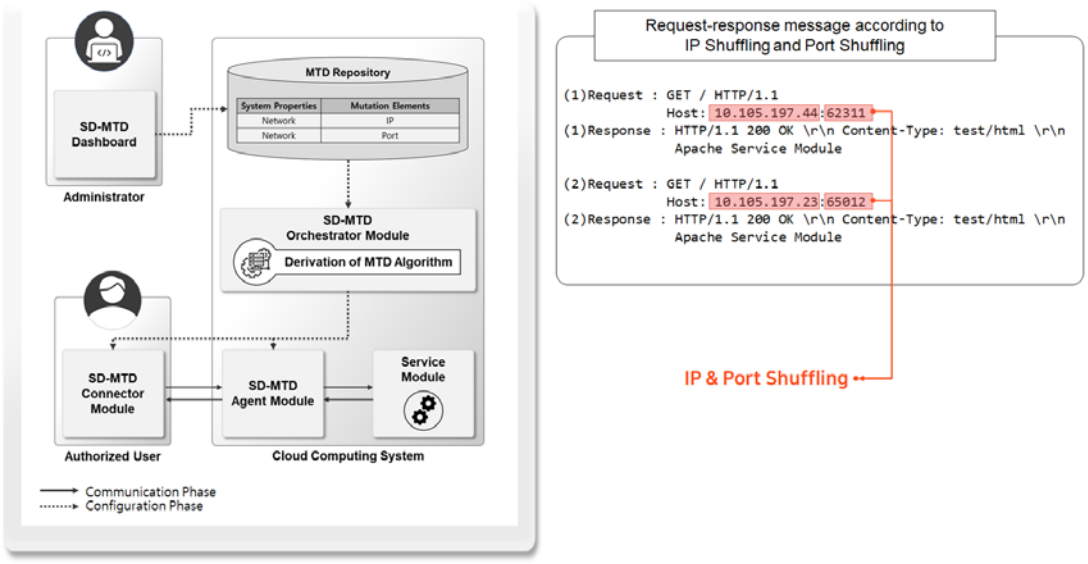

Fig. 6. IP and port shuffling applied to the Apache service

5 Conclusion

In recent years, container techniques have been integrated into cloud computing environments to maximize their efficiency, flexibility, and economic feasibility. Cloud computing systems, which offer a high agility and flexibility, have enabled the implementation of various MTD techniques. However, an analysis of existing MTD techniques revealed that although each technique was capable of deceiving attackers, they had limitations when applied to an actual cloud-computing system. These include wasted resources, management complexity caused by the additional function implementation and system introduction, and an increase in attack complexity. The proposed SD-MTD system was composed of an SD-MTD dashboard and SD-MTD orchestrator, SD-MTD agent, and SD-MTD connector modules. The administrator first selected the MTD for the service to be protected from the SD-MTD dashboard, and then defined its valid mutation range and mutation cycle. The SD-MTD orchestrator module verified that a valid range of mutation elements had been defined through the dashboard and created a mutation list. The SD-MTD agent and SD-MTD connector modules were located between the services provided by the cloud computing system and authorized service user. They obfuscated the communication according to the mutation list transmitted from the SD-MTD orchestrator module and the mutation cycle configured from the SD-MTD dashboard. The limitations of existing MTD techniques were derived prior to designing and implementing the SD-MTD system proposed in this study. Subsequently, the SD-MTD system requirements were derived, and the proposed system was designed and implemented according to these requirements. First, to achieve a flexible management, the system was designed to enable the administrator to monitor the system-resource status in real time through the implemented dashboard and configure a valid mutation range and mutation cycle for the MTD techniques.

Second, when multiple MTD techniques are implemented, the economic feasibility of the cloud computing system is compromised because each MTD technique has different requirements. To address this, a software-defined technique was employed to flexibly reflect the requirements. Third, an MTD has some limitations when faced with an increased attack complexity because of limited system resources. This limitation can be resolved by applying multiple MTD techniques. The SD-MTD system implemented in this study is expected to flexibly apply and manage both existing and future MTD techniques. In a follow-up study, we intend to apply context-aware techniques to control the MTD and automate the application according to the service security level by situation.

References

[1] C. Pahl, A. Brogi, J. Soldani and P. Jamshidi, "Cloud Container Technologies: A State-of-the-Art Review," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 677-692, 1 July-Sept. 2019. Article (CrossRef Link)

[2] Z. Kozhirbayev and R. O. Sinnott, "A performance comparison of container-based technologies for the cloud," *Future Generation Computer Systems*, 68, 175-182, 2017. Article (CrossRef Link)

[3] S. He, L. Guo, Y. Guo, C. Wu, M. Ghanem et al., "Elastic Application Container: A Lightweight Approach for Cloud Resource Provisioning," in *Proc. of 2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, pp. 15-22, 2012. Article (CrossRef Link)

[4] H. Jin, Z. Li, D. Zou, B. Yuan, "DSEOM: A Framework for Dynamic Security Evaluation and Optimization of MTD in Container-Based Cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 3, pp. 1125-1136, 1 May-June 2021. Article (CrossRef Link)

[5] X. Gao, Z. Gu, M. Kayaalp, D. Pendarakis and H. Wang, "ContainerLeaks: Emerging Security Threats of Information Leakages in Container Clouds," in *Proc. of 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 237-248, 2017. Article (CrossRef Link)

[6] X. Xu, H. Yu and X. Pei, "A Novel Resource Scheduling Approach in Container Based Clouds," in *Proc. of 2014 IEEE 17th International Conference on Computational Science and Engineering*, pp. 257-264, 2014. Article (CrossRef Link)

[7] A. Chung, J. Park, and G. Ganger, "Stratus: cost-aware container scheduling in the public cloud," in *Proc. of the ACM Symposium on Cloud Computing (SoCC '18). Association for Computing Machinery*, New York, NY, USA, 121–134, 2018. Article (CrossRef Link)

[8] W. Peng, F. Li, C. -T. Huang and X. Zou, "A moving-target defense strategy for Cloud-based services with heterogeneous and dynamic attack surfaces," in *Proc. of 2014 IEEE International Conference on Communications (ICC)*, pp. 804-809, 2014. Article (CrossRef Link)

[9] H. Alavizadeh, J. Jang-Jaccard and D. S. Kim, "Evaluation for Combination of Shuffle and Diversity on Moving Target Defense Strategy for Cloud Computing," in *Proc. of 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pp. 573-578, 2018. Article (CrossRef Link)

[10] A. Shaer, Ehab, Q. Duan, and J. Jafarian, "Random host mutation for moving target defense," in *Proc. of International Conference on Security and Privacy in Communication Systems*, Springer, Berlin, Heidelberg, pp. 310-327, 2012. Article (CrossRef Link)

[11] C. Lei, H. Zhang, J. Tan, Y. Zhang, X. Liu, "Moving Target Defense Techniques: A Survey," *Security and Communication Networks*, vol. 2018, Article ID 3759626, 25 pages, 2018. Article (CrossRef Link)

[12] P. Kampanakis, H. Perros and T. Beyene, "SDN-based solutions for Moving Target Defense network protection," in *Proc. of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pp. 1-6, 2014. Article (CrossRef Link)

[13] E. Al-Shaer, "Toward network configuration randomization for moving target defense," *Moving Target Defensem*, Springer, New York, NY, 153-159, 2011. Article (CrossRef Link)

[14] T. E. Carroll, M. Crouse, E. W. Fulp and K. S. Berenhaut, "Analysis of network address shuffling as a moving target defense," in *Proc. of 2014 IEEE International Conference on Communications (ICC)*, pp. 701-706, 2014. Article (CrossRef Link)
[15] J. Haadi, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: transparent moving target defense using software defined networking," in *Proc. of the first workshop on Hot topics in software defined networks*, pp. 127-132, 2012. Article (CrossRef Link)
[16] P. Dawson, and A. Butler, "IT Market Clock for Server Technology and SDx, 2014," Gartner Report 2014. 9.
[17] A. Gupta, L. Vanbever, M. Shahbaz, S. Donovan, B. Schlinker et al., "Sdx: A software defined internet exchange," *ACM SIGCOMM Computer Communication Review*, 44.4, 551-562, 2014. Article (CrossRef Link)
[18] A. Darabseh, M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, M. Vouk and A. Rindos, "SDDC: A Software Defined Datacenter Experimental Framework," in *Proc. of 2015 3rd International Conference on Future Internet of Things and Cloud*, pp. 189-194, 2015. Article (CrossRef Link)
[19] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown, "Where is the debugger for my software-defined network?," in *Proc. of the first workshop on Hot topics in software defined networks (HotSDN '12)*, *Association for Computing Machinery*, New York, NY, USA, 55–60, 2012. Article (CrossRef Link)
[20] A. Voellmy, and J. Wang, "Scalable software defined network controllers," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 289-290, 2012. Article (CrossRef Link)
[21] F. Chong, "National cyber leap year summit 2009: Co-chairs' report," *NITRD Program*, 2009.
[22] J. Cho, D. Sharma, H. Alavizadeh, S. Yoon, B. Noam et al, "Toward proactive, adaptive defense: A survey on moving target defense," *IEEE Communications Surveys & Tutorials*, 22.1, 709-745, 2020. Article (CrossRef Link)
[23] M. Green, "Characterizing network-based moving target defenses," in *Proc. of the Second ACM Workshop on Moving Target Defense*, pp. 31-35, 2015. Article (CrossRef Link)
[24] H. Okhravi et al., "Survey of cyber moving target techniques," Massachusetts Inst of Tech Lexington Lincoln Lab, 2018. Available: https://apps.dtic.mil/sti/pdfs/AD1055276.pdf
[25] B. Hong, and D. Kim, "Assessing the effectiveness of moving target defenses using security models," *IEEE Transactions on Dependable and Secure Computing*, 13.2, 163-177, 2016. Article (CrossRef Link)
[26] A. Alshamrani, S. Myneni, A. Chowdhary, D. Huang, "A Survey on Advanced Persistent Threats: Techniques, Solutions, Challenges, and Research Opportunities," *IEEE Communications Surveys & Tutorials*, Vol. 21, no. 2, pp. 1851-1877, Secondquarter 2019. Article (CrossRef Link)
[27] Y. -B. Luo, B. -S. Wang, X. -F. Wang, X. -F. Hu, G. -L. Cai and H. Sun, "RPAH: Random Port and Address Hopping for Thwarting Internal and External Adversaries," in *Proc. of 2015 IEEE Trustcom/BigDataSE/ISPA*, pp. 263-270, 2015. Article (CrossRef Link)
[28] J. Park, Y. Lee, K. Kang, S. Lee, and K. Park, "Ghost-MTD: Moving Target Defense via Protocol Mutation for Mission-Critical Cloud Systems," *Energies*, 13.8, 1883, 2020. Article (CrossRef Link)
[29] Y. Huang, and A. Ghosh, "Introducing diversity and uncertainty to create moving attack surfaces for web services," *Moving target defense*, Springer, New York, NY, 131-151, 2011. Article (CrossRef Link)
[30] M. Taguinod, A. Doupé, Z. Zhao and G. Ahn, "Toward a Moving Target Defense for Web Applications," in *Proc. of* 2015 *IEEE International Conference on Information Reuse and Integration*, pp. 510-517, 2015. Article (CrossRef Link)

Ki-Wan Kang received the B.S. degree in information security engineering from Soonchunhyang University, South Korea, in 2019, and the M.S degree in information security engineering from Sejong University, South Korea, in 2021. His research interests include cloud computing security and system security.

Jung Take Seo received the M.S. degree in Computer Engineering from Ajou University, South Korea, in 2001, and the Ph.D. degree in Information Security Engineering from Korea University, South Korea, in 2006. He worked for National Security Research Institute as a senior researcher. He is currently an Associate Professor with the Department of Computer Engineering, Gachon University. His research interests include CPS security, ICS cybersecurity, smart grid security, nuclear power plant security, smart factory security, smart city security, and automotive cybersecurity.

Sung Hoon Baek received the B.S. degree in electronics engineering from Kyungpook National University, Korea, in 1997, the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 1999, the Ph.D. degree in electrical engineering from KAIST in 2008. He worked for Electronics Telecommunication Research Institute (ETRI) as an R&D staff from 1999 to 2005 and for Samsung Electronics as a senior R&D staff from 2008 to 2011. He has been an associate professor in the department of computer engineering at Jungwon University since 2011. His research interests include storage system, operating system, security issues in mobile computing systems, and particulate matter monitoring.

Chulwoo Kim received the B.S. degree in computer science from Yonsei University, South Korea, in 2005, the M.S. degree in computer science from Pace University, New York, in 2018. He is currently working for LG CNS America as a senior data engineer. His research interests include computer vision, big data, and machine learning.

Ki-Woong Park received the B.S. degree in computer science from Yonsei University, South Korea, in 2005, the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 2007, and the Ph.D. degree in electrical engineering from KAIST in 2012. He received a 2009–2010 Microsoft Graduate Research Fellowship. He worked for National Security Research Institute as a senior researcher. He has been a professor in the department of computer and information security at Sejong University. His research interests include security issues for cloud and mobile computing systems as well as the actual system implementation and subsequent evaluation in a real computing system.