energies

Article

Ghost-MTD: Moving Target Defense via Protocol Mutation for Mission-Critical Cloud Systems

Jun-Gyu Park [1], Yangjae Lee [1], Ki-Wan Kang [1], Sang-Hoon Lee [2] and Ki-Woong Park [3,*]

1 SysCore Laboratory, Sejong University, Seoul 05006, Korea; wnsrb3001@gmail.com (J.-G.P.); leelambjae@gmail.com (Y.L.); kkwan0226@gmail.com (K.-W.K.)
2 Agency for Defense Development, Daejeon 34060, Korea; shlee@add.re.kr
3 Department of Computer and Information Security, Sejong University, Seoul 05006, Korea
* Correspondence: woongbak@sejong.ac.kr; Tel.: +82-2-6935-2453

Received: 11 March 2020; Accepted: 10 April 2020; Published: 13 April 2020

Abstract: Research on various security technologies has been actively underway to protect systems from attackers. However, attackers can secure enough time to reconnoiter and attack the target system owing to its static nature. This develops asymmetric warfare in which attackers outwit defenders. Moving target defense (MTD) technologies, which obfuscate the attack surface by modifying the main properties of the potential target system, have been gaining attention as an active cyber security technology. Particularly, network-based MTD (NMTD) technologies, which dynamically mutate the network configuration information, such as IP and ports of the potential target system, can dramatically increase the time required for an attacker to analyze the system. Therefore, this system defense technology has been actively researched. However, increasing the analysis complexity of the target system is limited in conventional NMTD because the variation of system properties (e.g., IP, port) that can be mutated is restricted by the system configuration environment. Therefore, there is a need for an MTD technique that effectively delays an attacker during the system analysis by increasing the variation of system properties. Additionally, in terms of practicality, minimizing the computational overhead arising by the MTD technology and solving the compatibility problem with existing communication protocols are critical issues that cannot be overlooked. In this study, we propose a technology called Ghost-MTD (*gMTD*). *gMTD* allows only the user who is aware of protocol mutation patterns to correctly communicate with the service modules of the server system through protocol mutation using the pre-shared one-time bit sequence. Otherwise, *gMTD* deceives the attackers who attempt to infiltrate the system by redirecting their messages to a decoy-hole module. The experimental results show that the proposed technology enables protocol mutation and validation with a very low performance overhead of only 3.28% to 4.97% using an m-bit ($m \geq 4$) length one-time bit sequence and can be applied to real systems regardless of the specific communication protocols.

Keywords: moving target defense; deception; protocol mutation; mission-critical cloud systems

1. Introduction

Sophisticated cyber-attacks, such as advanced persistent threats, called APT [1–3], have been on the rise. Research on security technologies has been actively conducted to protect systems from such highly sophisticated cyber-attacks [4,5]. However, attackers can secure enough time to reconnoiter and attack the target system owing to the static nature of the target system [6,7], developing an asymmetric warfare relationship between attackers and defenders.

Much research on system defense mechanisms has been carried out to devise methods of rebalancing this asymmetric warfare relationship. The moving target defense (MTD) technology has been proposed to respond to these threats [8]. MTD is an active security technology that blocks various cyber-attacks in advance by proactively modifying the major system properties [9,10]. Network-based MTD (NMTD), which is a type of MTD technology, can dramatically increase the complexity of the system analysis by continuously modifying the network property information [11]. For example, the network address mutation technology dynamically mutates the network configuration information, such as IP and ports. This is one of the system defense technologies most actively studied because it can dramatically increase the time required for attackers to analyze the system [12]. Jafarian proposed an openflow-random host mutation (OF-RHM) that periodically modifies the virtual IP address assigned to the system to be protected [13], thus dramatically increasing the time required for attackers to analyze the system. However, OF-RHM can only be applied to systems with a virtual network technology since it is implemented based on the software-defined network technology. Luo proposed random port and address hopping (RPAH) that predefines the change cycle of IP addresses and port information applied to the system and randomly mutates them according to the change cycle [14]. In RPAH, the gateway creates the virtual IPs and dynamically applies them to the system. Clients transmit packets using the virtual IP address rather than the real IP address of the system. The real IP address and port information of the system are retained, while packets are routed by converting the virtual IP address and port numbers assigned to the system at the relevant time into the real IP address and port numbers. Accordingly, messages from attackers using invalid IP addresses are detected and rejected. Dunlop proposed the moving target IPv6 defense (MT6D) framework, which mutates 64-bit interface identifiers in an IPv6 environment [15]. MT6D generates an IPv6 address to be mutated through a hash function based on a pre-shared time period between senders and receivers and mutates the current IPv6 address into a newly generated IPv6 address.

However, NMTD has a limitation in increasing the complexity against attacks since the variation of system properties (e.g., IP address and port information) that it can mutate is restricted depending on the system configuration environment. To overcome this limitation, there is a need for an MTD technique that more effectively delays the system analysis by increasing the variation of system properties. Additionally, in terms of practicality, minimizing the computational overhead arising by the MTD technology and solving the compatibility problem with the existing communication protocols are critical issues that cannot be overlooked [16,17]. In this study, we propose a technology called Ghost-MTD (*gMTD*) that allows only the user who is aware of protocol mutation patterns to correctly communicate with the service modules of the server system through protocol mutation using the pre-shared one-time bit sequence (OTBS). Otherwise, *gMTD* deceives the attackers who attempt to infiltrate the system by redirecting their messages to the decoy-hole module [18,19]. In *gMTD*, the OTBS was designed inspired by the OTP mechanism [20,21] and refers to a bit sequence used to mutate the communication protocols between the user and the server system. The OTBS is generated separately for each user during user registration and is shared between the user and the server system. For example, $OTBS_{ID}^{n}$ is the OTBS used in the n-th communication between the user with a specified ID and the server system. As shown in Figure 1, in the n-th communication, an m-bit $OTBS_{ID}^{n}$ is extracted from the OTBS shared between the user and the server system, and the extracted data is used to modify the protocol for the user-server communication. In this way, *gMTD* enables a one-time protocol mechanism by continuously modifying the protocols for the communication between the user and the server system. Through this, *gMTD* only allows the user who is aware of protocol mutation patterns to communicate with the service modules of the server system correctly, and deceives the attackers who attempt to infiltrate the system by redirecting all messages from attackers to the decoy-hole module.

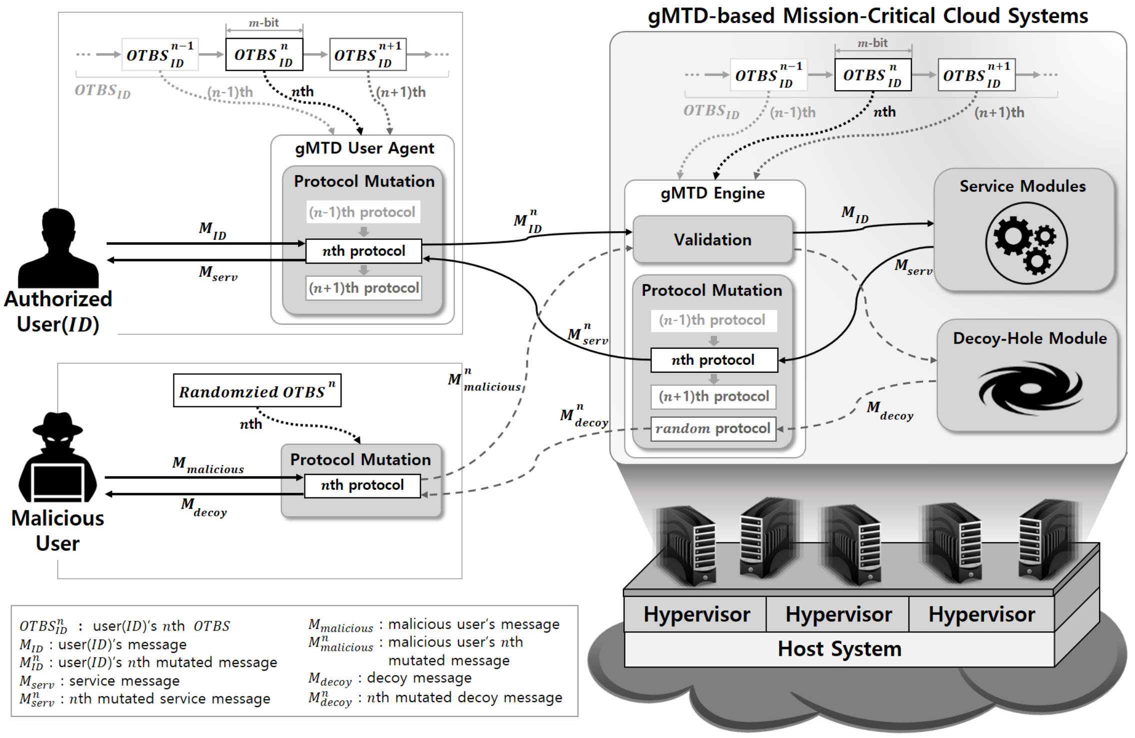

gMTD-based Mission-Critical Cloud Systems
m-bit
OTBS$_{ID}^{n-1}$ OTBS$_{ID}^{n}$ OTBS$_{ID}^{n+1}$
OTBS$_{ID}$
(n-1)th nth (n+1)th
gMTD User Agent
Protocol Mutation
(n-1)th protocol
nth protocol
(n+1)th protocol
Authorized User(ID)
M_{ID}
M_{serv}
M_{ID}^{n}
M_{serv}^{n}
Randomzied OTBS^{n}
Protocol Mutation
nth protocol
Malicious User
$M_{malicious}$
M_{decoy}
$M_{malicious}^{n}$
M_{decoy}^{n}
gMTD Engine
Validation
Protocol Mutation
random protocol
Service Modules
Decoy-Hole Module
Hypervisor
Host System
$OTBS_{ID}^{n}$: user(ID)'s nth OTBS
M_{ID} : user(ID)'s message
M_{ID}^{n} : user(ID)'s nth mutated message
M_{serv} : service message
M_{serv}^{n} : nth mutated service message
$M_{malicious}$: malicious user's message
$M_{malicious}^{n}$: malicious user's nth mutated message
M_{decoy} : decoy message
M_{decoy}^{n} : nth mutated decoy message

Figure 1. Overall architecture of Ghost moving target defense (*gMTD*) and an example of a mission-critical cloud system enhanced with *gMTD*.

The contribution of this paper can be summarized as follows.

- We devised a continuous protocol mutation scheme called *gMTD* to effectively delay the analysis of the system targeted by attackers. *gMTD* increases the variation of system properties through the continuous change of the protocols.
- We devised a protocol mutation scheme that allows only the user who is aware of protocol modification patterns to communicate with the service modules of the server system and the decoy-hole module-based attacker-deception scheme. We can validate protocol compliance with the pre-shared OTBS, and *gMTD* deceives the attackers who attempt to infiltrate the system by redirecting invalid protocol compliance messages to the decoy-hole module.
- We devised a cost-effective protocol-mutation scheme taking into account practicality and applicability. The experimental results show that *gMTD* enables a mutation protocol with very low performance overhead of at least 3.28% and 0.0086% per additional bit using an m-bit ($m \geq 4$) OTBS.

This paper is structured as follows: Section 2 describes in detail the architecture and protocol mutation of *gMTD* for mission-critical cloud systems. Section 3 evaluates the implemented prototype, and Section 4 concludes this paper with future research direction.

2. System Internal of *gMTD*

In this section, we describe the overall architecture of *gMTD* and its core technologies; i.e., the mechanism for protocol mutation and validation scheme (see Sections 2.1 and 2.2, respectively). The OTBS used for protocol mutation in the proposed technology is assumed to be generated differently for each user during the user registration process and to have a mechanism securely shared between the user and the server system. For example, the user's OTBS with ID is $OTBS_{ID}$ and the user's n-th OTBS is $OTBS_{ID}^{n}$. As shown in Figure 1, in the n-th communication between the user with ID and the server system, the server system and the user generate protocols using an m-bit $OTBS_{ID}^{n}$ extracted from $OTBS_{ID}$. Furthermore, the message used for protocols is mutated using a different

$OTBS_{ID}^{n}$ at each time during user-server communication. The user sends a message appending his ID to the server system. Then, the server system validates the received messages using the $OTBS_{ID}^{n}$ extracted from the same $OTBS_{ID}$ that is shared with the user. The server system redirects incoming messages to different modules according to the validation result. If the validation result is positive, the message is redirected to the service modules; otherwise, it is redirected to the decoy-hole module. When communicating with the user, the response to valid messages generated by the service modules are mutated using the same $OTBS_{ID}^{n}$. Contrarily, the response to invalid messages generated by the decoy-hole module are mutated using a randomly generated $OTBS_{decoy}^{n}$. The user recovers the message received from the server system using the $OTBS_{ID}^{n}$. The protocol mutation scheme in *gMTD* consists of split-and-swap and insertion operations, as shown in Figure 2. The split-and-swap operation splits the original message using the first 4 bits of $OTBS_{ID}^{n}$ for protocol mutation and swaps the split messages. This operation can be performed by one swap instruction provided by the native CPU, thus enabling light-weight protocol mutation.

After the split-and-swap operation, the insert operation is performed to scatter the message to be sent using the *m*-bit $OTBS_{ID}^{n}$. This allows the server system to validate protocol compliance for the received messages with only partial validation operation, thus enabling protocol mutation that provides computational efficiency and ease of validation.

2.1. Protocol Mutation Scheme

This subsection describes the proposed protocol mutation scheme on the basis of *gMTD*. As shown in Figure 2, by implementing split-and-swap and insertion operations, we have designed a cost-effective protocol mutation technology that enables light-weight protocol mutation. The split-and-swap operation splits the user's original message, M_{ID}, into j (=MSB 4 bits of $OTBS_{ID}^{n}$) pieces of split messages. Then, the split messages are further split in half and swapped repeatedly until their final sizes become 1 bit. Finally, M_{ID} is converted into M_{ID}^{s}. The insert operation splits the $OTBS_{ID}^{n}$ bit-by-bit and inserts each split bit into the M_{ID}^{s} at regular intervals (=LSB 4 bits of $OTBS_{ID}^{n}$). Consequently, the original message M_{ID}^{s} is mutated to M_{ID}^{n}. Before M_{ID}^{n} is sent, the last index of the message where $OTBS_{ID}^{n}$ was inserted and the user's ID are added to the end of M_{ID}^{n}. In protocol mutation, the swap operation is performed by only one instruction using the swap command provided by the native CPU, and does not require any additional memory space. This approach is computationally efficient because it can reduce the CPU and memory overhead consumption required by the user and the server system. Additionally, the insertion operation enables the server system to validate protocol compliance with only partial validation of the received message, resulting in quick message validation. A detailed description of the split-and-swap and insertion operations is as follows.

- Split-and-swap: It splits the message (=M_{ID}) to be transmitted using the *MSB[4]* (=MSB 4 bits of $OTBS_{ID}^{n}$) and swaps the split messages. The *MSB[4]* are used as the number of subsections in which the message is initially split. The split-and-swap operation consists of r rounds. Before the start of each round, the original message M_{ID} is split into *MSB[4]* pieces. For example, if the first 4 bits of $OTBS_{ID}^{n}$ are 16, the 128-bit message is initially split into 16 subsections of 8 bits. In each round, the split message is further split in half and swapped mutually. After the final iteration, the size of the split messages is 1 bit. The number of initially split subsections must be an even number for ease of the split-and-swap operation. Therefore, if the last 4 bits of the $OTBS_{ID}^{n}$ are an odd number, 1 is added to make it even. Through this operation, M_{ID} is converted to M_{ID}^{s}. The server system can invert the message through the same split-and-swap operation.
- Insertion: It splits the $OTBS_{ID}^{n}$ bit-by-bit and sequentially inserts each split bit into the M_{ID}^{s} at regular intervals set to *LSB[4]* (=LSB 4 bits of $OTBS_{ID}^{n}$). The insert operation consists of m rounds of operations. Since the index value indicating the position where the $OTBS_{ID}^{n}$ is to be inserted is calculated through a mod operation with message length k, the index value does not exceed the maximum value of the message length. After round m, the index value indicating the position where the $OTBS_{ID}^{n}$ was lastly inserted and the user's ID are appended to the message to undergo

validation by the server system. Therefore, M^s_{ID} is converted into M^n_{ID}, which corresponds to the message to be sent to the server system. This insertion operation enables the server system to quickly validate the message because invalid messages are detected if any of the $OTBS^n_{ID}$ scattered in the message is incorrect.

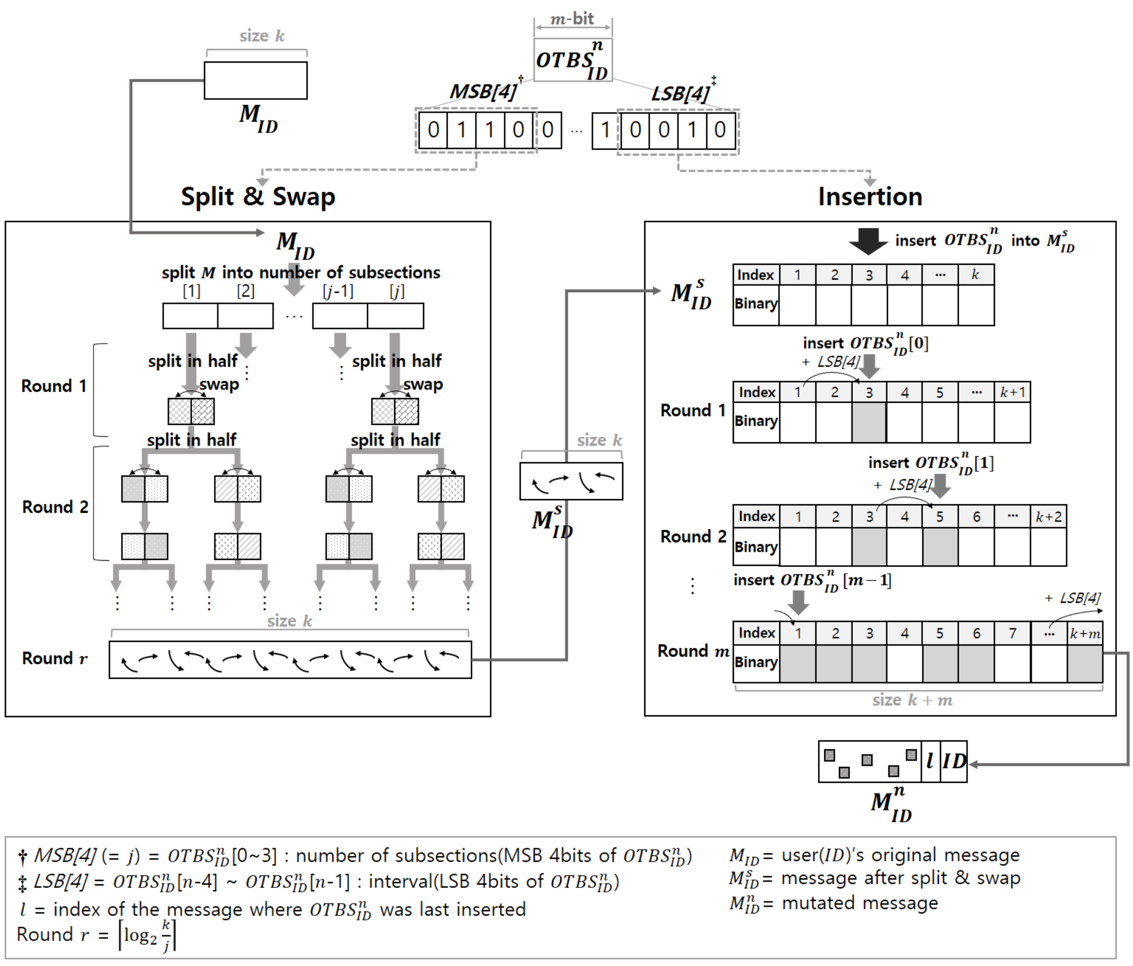

size k
m-bit
$OTBS^n_{ID}$
M_{ID}
$MSB[4]^\dagger$
$LSB[4]^\ddagger$
0 1 1 0 0 ··· 1 0 0 1 0
Split & Swap
Insertion
M_{ID}
split M into number of subsections
[1] [2] [j-1] [j]
split in half
swap
Round 1
Round 2
Round r
size k
M^s_{ID}
insert $OTBS^n_{ID}$ into M^s_{ID}
Index 1 2 3 4 ··· k
Binary
insert $OTBS^n_{ID}[0]$
+ LSB[4]
Index 1 2 3 4 5 ··· k+1
insert $OTBS^n_{ID}[1]$
Index 1 2 3 4 5 6 ··· k+2
insert $OTBS^n_{ID}[m-1]$
Round m
Index 1 2 3 4 5 6 7 ··· k+m
size $k+m$
l ID
M^n_{ID}
† $MSB[4]$ (= j) = $OTBS^n_{ID}[0\sim3]$: number of subsections(MSB 4bits of $OTBS^n_{ID}$)
‡ $LSB[4]$ = $OTBS^n_{ID}[n\text{-}4]$ ~ $OTBS^n_{ID}[n\text{-}1]$: interval(LSB 4bits of $OTBS^n_{ID}$)
l = index of the message where $OTBS^n_{ID}$ was last inserted
Round r = $\lceil \log_2 \frac{k}{j} \rceil$
M_{ID} = user(ID)'s original message
M^s_{ID} = message after split & swap
M^n_{ID} = mutated message

Figure 2. Internal operations overview of protocol mutation.

2.2. Message Validation and Deception

This section describes how the server system verifies the protocol compliance for the incoming messages and how it deceives attackers. As shown in Figure 3, we have designed a computationally efficient message validation and attacker deception mechanism. Because the server system must only allow a user who is aware of the protocol mutation patterns to correctly communicate with the service modules of the server system, the server system has to validate the messages that users have mutated through the protocol mutation process using the correct $OTBS^n_{ID}$. The proposed scheme can detect invalid messages if any of the $OTBS^n_{ID}$ scattered in the message is incorrect. In the n-th communication between the server system and the user, the former extracts the user's ID at the end of the M^n_{ID} and validates the M^n_{ID} using the pre-shared $OTBS^n_{ID}$. Moreover, the server system extracts the index l of the message where the $OTBS^n_{ID}$ was last inserted from the M^n_{ID} and uses l to validate the M^n_{ID}. The server system inverts the M^n_{ID} into the M^s_{ID} by extracting the $OTBS^n_{ID}$ inserted in the M^n_{ID} in reverse order, as a message validation operation. If the $OTBS^n_{ID}$ inserted in the message is invalid, the server system immediately stops validation and redirects the received messages to the decoy-hole module. Then, the response messages generated by the decoy-hole module are mutated using a randomly generated $OTBS^n_{decoy}$. If the $OTBS^n_{ID}$ inserted in the M^n_{ID} is valid, the server system inverts the M^s_{ID} into the

M_{ID} through the split-and-swap operation and sends the correct response message to the user by redirecting it to the service modules.

Unlike the native message validation scheme, the proposed scheme enables the detection of invalid messages without performing the entire reverse operations of protocol mutation. Therefore, *gMTD* promptly and computationally efficiently validates the protocol compliance of incoming messages.

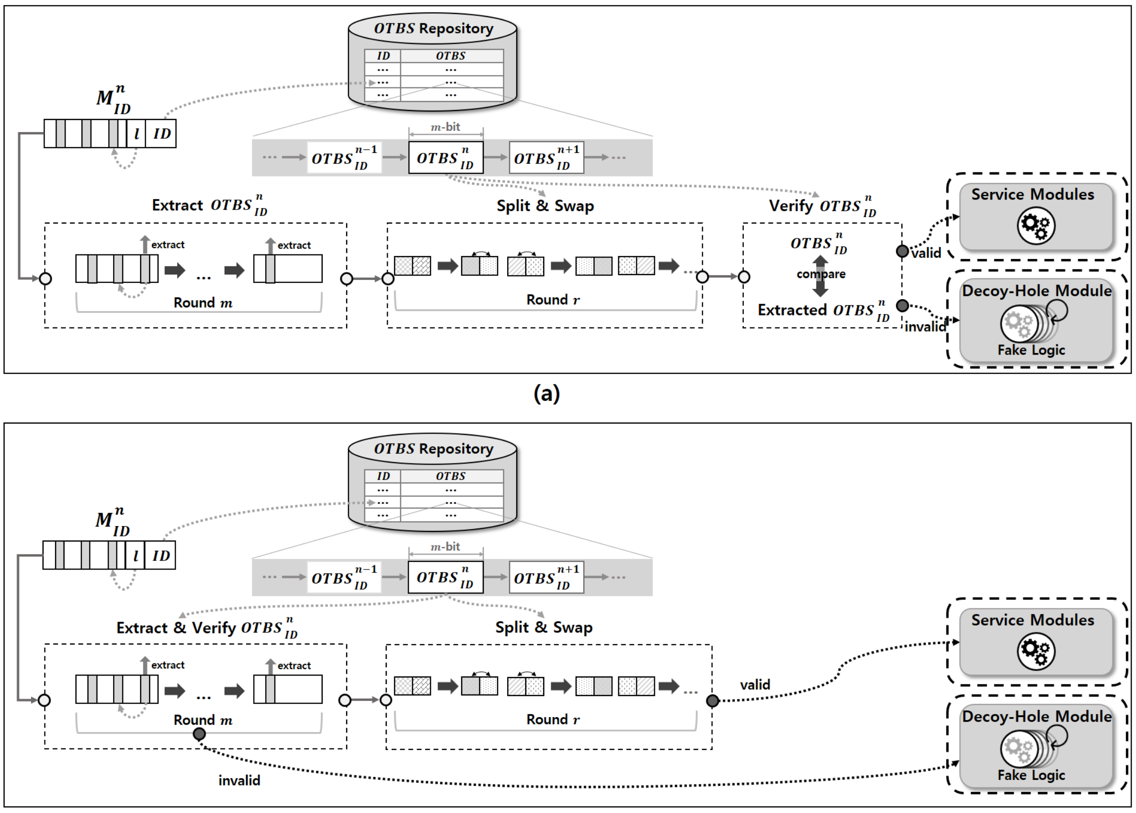


Figure 3. Operation process for message validation and deception: (**a**) native validation scheme; (**b**) validation scheme on the basis of *gMTD*.

2.3. Decoy-Hole Module

gMTD deceives attackers who attempt to infiltrate the system by redirecting their messages to a decoy-hole module. Consequently, the decoy-hole module is equipped with a fake system logic, which generates a fake response to an attacker. As shown in Figure 3a,b, all system logic and data of the decoy-hole module are completely separate from the service module to protect the service module. In addition, the fake system logic inside the decoy-hole module is periodically changed to another service to prevent attackers from understanding the internal logic used by the decoy-hole module. In this way, *gMTD* generates and sends response messages to attackers' requests through the decoy-hole module, thereby deceiving attackers who attempt to infiltrate the system and increasing the time required by attackers to analyze the system.

In this way, *gMTD* generates and sends the response messages as the response of the attackers' requests through the decoy-hole module, thereby deceives the attackers who attempt to infiltrate the system and increase the time required for attackers to analyze the system.

3. Experiment

3.1. Operation Overhead

We measured the overhead arising from operation and memory consumption to perform message validation in *gMTD*. We conducted an experiment on a server equipped with Intel(R) CoreTM i7-8700 and 32 GB RAM. For this experiment, we created user and server system virtual machines (VMs) equipped with 1 vcpu, 1 GB memory, and Ubuntu 18.04 OS. To measure the overhead required for protocol mutation in *gMTD*, we measured the CPU and memory utilization required for protocol mutation and message validation by changing the length of the $OTBS^n_{ID}$ from 4 to 200. The result of the performance measurement showed that the average CPU usage increase rate added for protocol mutation and message validation using the 4-bit and 200-bit long $OTBS^n_{ID}$ was 0.009% and 0.03%, respectively. Additionally, the average memory usage was 168 kB regardless of the length of the $OTBS^n_{ID}$. The results verify that the proposed scheme can realize protocol mutation-based MTD with very low computation and memory overhead.

3.2. Validation Efficiency

We conducted two experiments to evaluate the effectiveness of message validation in *gMTD*. In the first experiment, we compared the validation process time of the native message validation scheme, which performs the entire reverse protocol mutation operation, with that of *gMTD*, which verifies invalid messages through sequential reading. We generated an M^n_{ID} by scattering the 100-bit $OTBS^n_{ID}$ in the message. Then, we calculated the computation time spent on the invalid message by changing the number of bits of the valid $OTBS^n_{ID}$ from 10 to 100. Because the $OTBS^n_{ID}$ is validated in reverse order from the last $OTBS^n_{ID}$ inserted in the message, it is inserted into the message to be validated. For example, if the number of valid sequential $OTBS^n_{ID}$ bits is 10, an invalid 90-bit $OTBS^n_{ID}$ is inserted into the message first, and the valid 10-bit $OTBS^n_{ID}$ is inserted afterward. Figure 4 shows the comparison of the message validation process times. The *gMTD* immediately stops the reverse protocol mutation operation required for message validation when the $OTBS^n_{ID}$ inserted in the incoming messages were invalid. The native message validation scheme always requires more time for validation compared with the validation scheme in *gMTD* that promptly and efficiently validates the protocol compliance of the message.

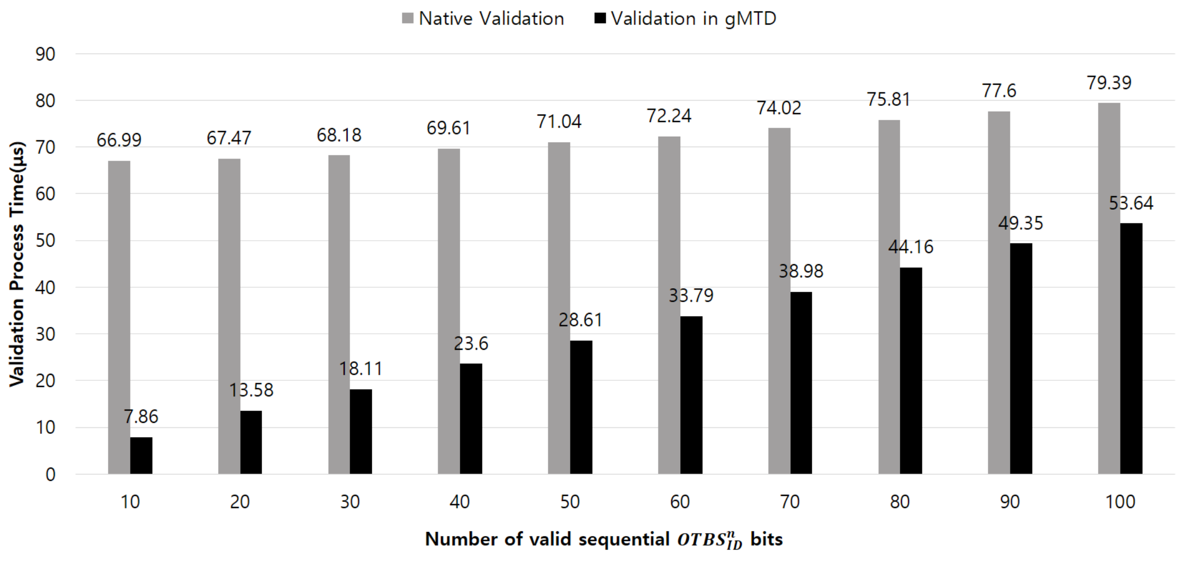


Figure 4. Measuring the validation process time with varying number of valid sequential $OTBS^n_{ID}$ bits.

Second, we compared the operation time of message validation between the native message validation scheme and the validation scheme in *gMTD* in terms of message length. In this experiment, the number of bits to be scattered in the message was fixed at 100 bits, and the validation process time for valid message validation was measured by changing the message length from 4 to 1024 bits. Figure 5 shows the dependence of the validation process time on the message length in the compared validation schemes. The results show that the validation process times of both validation schemes slightly increased with increasing message length. This indicates that *gMTD* can validate the protocol compliance of messages without being significantly influenced by the message length.

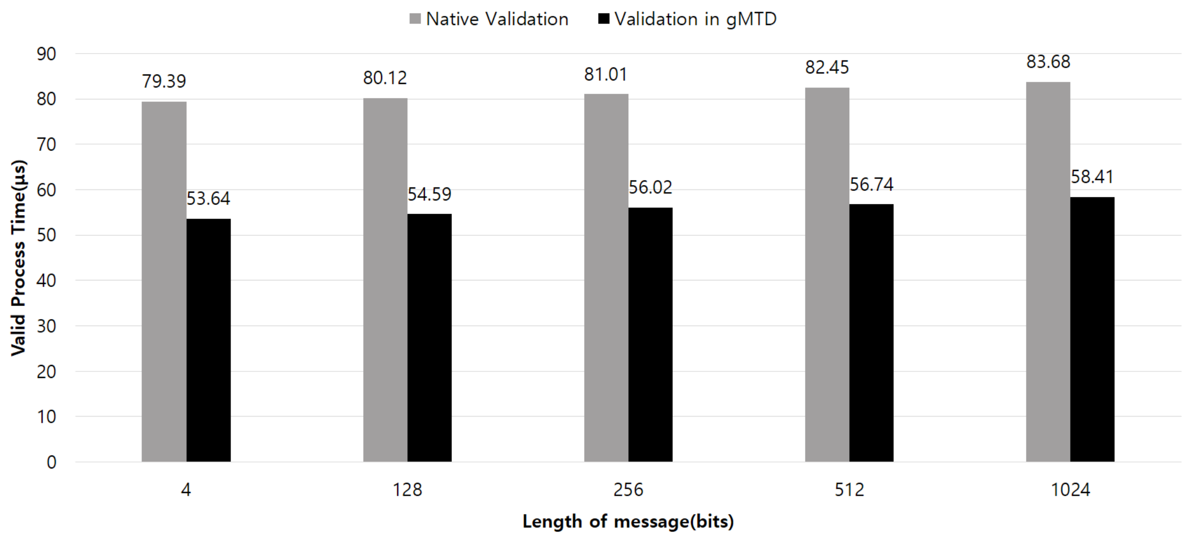


Figure 5. Measuring the validation process time with varying message length.

3.3. Performance Overhead

To evaluate the performance overhead of the proposed technology, we compared non-*gMTD* and *gMTD* in an HTTP service environment. The transaction per second (TPS) was measured with various $OTBS^n_{ID}$ lengths. Figure 6 shows the performance comparison. The measured TPS of non-*gMTD* in which $OTBS^n_{ID}$ was not used is the same for all $OTBS^n_{ID}$ lengths. The TPS of non-*gMTD* for every bit was 368.12. The TPS for *gMTD* was 356.01 for 4 bits and 349.81 for 200 bits. The result shows that *gMTD* generates only 3.28%–4.97% of overhead compared to non-*gMTD*, and demonstrates an overhead increase of 0.0086% for every 1-bit increase in the $OTBS^n_{ID}$. This indicates that user messages are very efficiently validated and redirected to the service modules. Additionally, *gMTD* has the advantage of being easily applied to real systems regardless of the specific communication protocols.

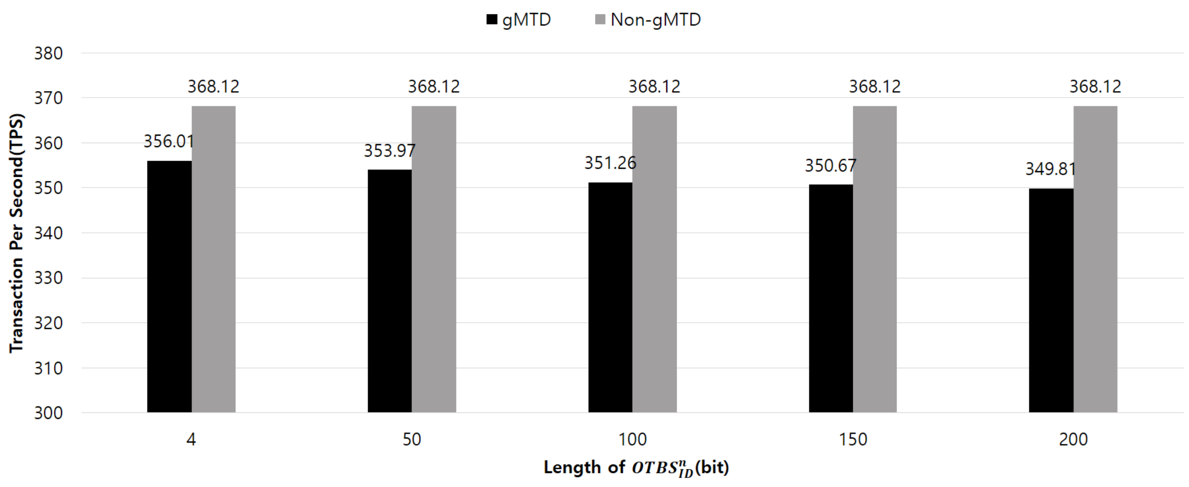


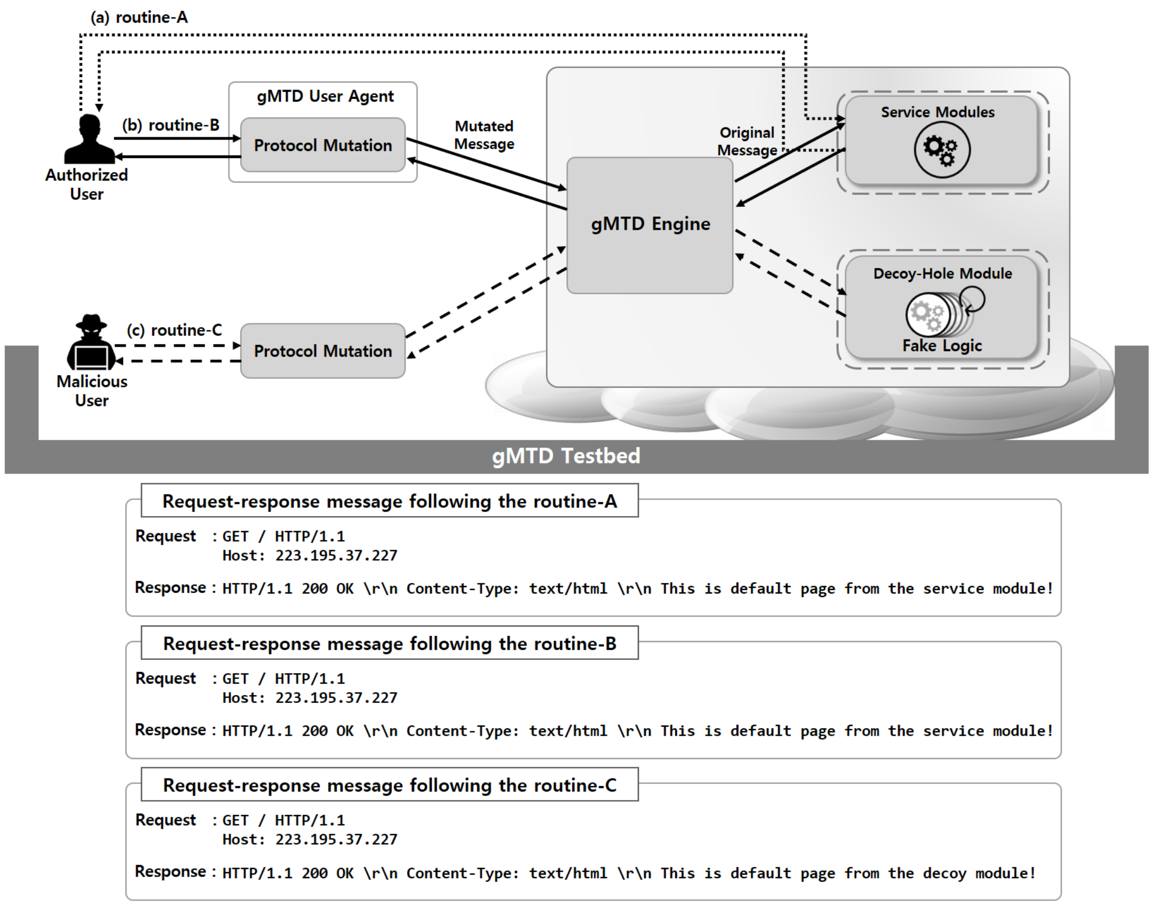
Figure 6. Measuring transaction per second (TPS) with varying length of $OTBS^n_{ID}$.

3.4. MTD Efficacy of gMTD

In this section, we evaluated the MTD efficacy for delaying an attacker's system analysis by comparing the number of system variation cases between NMTD and *gMTD*. In *gMTD*, the value of the OTBS, which is periodically changed, is utilized as a seed value to mutate the communication protocols between users and the server system. An attacker needs to perform brute-force trials to identify the current protocol pattern. Therefore, the MTD efficacy of *gMTD* depends on the length of $OTBS_{ID}^{n}$ because the protocol mutation pattern is derived from $OTBS_{ID}^{n}$. The maximum number of system variation cases in *gMTD* can be defined as 2^k(k = bits length of $OTBS_{ID}^{n}$). *gMTD* allows the system variation width to change flexibly by configuring k. Therefore, the appropriate length of $OTBS_{ID}^{n}$(=k) can be determined by considering the security requirements and the acceptable computational overhead (the overhead increases as the value of k increases). Further, the number of system variations in NMTD is determined by network properties, such as the size of the subnet mask and the number of ports owned by the operating institution and is expressed as $2^n \times p$ (n: size of subnet mask, p: number of ports). Therefore, the system variations for NMTD and *gMTD* can be compared using the values of n, p, and k. Note also that a more powerful moving target defense mechanism can be realized by coupling *gMTD* with conventional NMTD because *gMTD* requires no modifications to either the network or the application layer.

3.5. Implementation Correctness of gMTD

To verify the correctness of the *gMTD* implementation, we built a *gMTD* testbed, as shown in Figure 7. The testbed is designed to allow a virtual user to generate request–response transactions to and from a virtual service on the basis of *gMTD*, allowing us to determine whether the protocol mutation and the decoy-hole module are working correctly. In this experiment, we verified the implementation correctness of *gMTD* by sending a virtual request using the following three request–response routines and then analyzing the contents of the corresponding response messages. The first request–response routine is a routine in which a user directly accesses the service module without using *gMTD*, termed routine-A, as shown in Figure 7a. The second request–response routine is a *gMTD* routine in which an authorized user accesses the service module with a valid $OTBS_{ID}^{n}$, termed routine-B, as shown in Figure 7b. The third request–response routine is a routine where a malicious user accesses the service module using an incorrect $OTBS_{ID}^{n}$, termed routine-C, as shown in Figure 7c. If *gMTD* is implemented correctly, the response messages for routine-A and for routine-B must be the same, and both response messages must be sent from the service module. Conversely, the response message for routine-C must be transmitted from the decoy-hole module. In this experiment, the transactions following routine-A, routine-B, and routine-C were randomly generated to evaluate the algorithm, and it was confirmed that the algorithm performed correctly in all cases.



Request-response message following the routine-A

```
Request  : GET / HTTP/1.1
           Host: 223.195.37.227

Response : HTTP/1.1 200 OK \r\n Content-Type: text/html \r\n This is default page from the service module!
```

Request-response message following the routine-B

```
Request  : GET / HTTP/1.1
           Host: 223.195.37.227

Response : HTTP/1.1 200 OK \r\n Content-Type: text/html \r\n This is default page from the service module!
```

Request-response message following the routine-C

```
Request  : GET / HTTP/1.1
           Host: 223.195.37.227

Response : HTTP/1.1 200 OK \r\n Content-Type: text/html \r\n This is default page from the decoy module!
```

Figure 7. *gMTD* testbed for verification of the implementation correctness: (**a**) routine-A is a non-*gMTD* routine where a user directly accesses the service module without using *gMTD*; (**b**) routine-B is a *gMTD* routine in which an authorized user accesses to the service module on the basis of *gMTD*; (**c**) routine-C is a routine where a malicious user accesses to the service module.

4. Conclusions

In this study, we proposed *gMTD*, a novel technology that allows only the user who is aware of the protocol mutation pattern to communicate with the service modules of the server system and efficiently deceives attackers. Incoming messages are mutated by the protocol, which is generated using the pre-shared OTBS. Then, the server system validates the protocol compliance of the message using the same OTBS with the user. If the mutation patterns of the incoming messages are invalid, the server system deceives attackers by redirecting incoming messages to the decoy-hole module. The experimental results showed that the proposed MTD technology enables efficient protocol mutation and validation and redirection of incoming messages to the service modules. Moreover, *gMTD* can be easily applied to real systems regardless of the specific communication protocols. Nevertheless, in this study, we have designed and evaluated the technology in terms of cost-effective blocking of attack reconnaissance and attacker deception without considering the latency overhead. As a next step, we intend to account for latency overhead and are working towards a highly efficient protocol mutation MTD technology that defends against attacks effectively and proactively.

Author Contributions: Conceptualization, K.-W.P. and J.-G.P.; Formal analysis, J.-G.P. and K.-W.P.; Funding acquisition, K.-W.P.; Investigation, S.-H.L. and K.-W.P.; Methodology, K.-W.P.; Project administration, K.-W.P.; Resources, K.-W.P.; Software, J.-G.P., K.-W.K. and K.-W.P.; Supervision, K.-W.P.; Writing–review and editing, J.-G.P., Y.L. and K.-W.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Agency for Defense Development (ADD) project number UD180012ED.

Acknowledgments: This work was supported by Project No. UD180012ED.

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

References

1. Chen, P.; Desmet, L.; Huygens, C. A Study on Advanced Persistent Threats. In Proceedings of the IFIP International Conference on Communications and Multimedia Security, Aveiro, Portugal, 25–26 September 2014; pp. 142–149.
2. Yang, L.X.; Li, P.; Yang, X.; Tang, Y.Y. Security evaluation of the cyber networks under advanced persistent threats. *IEEE Access* **2017**, *5*, 20111–20123. [CrossRef]
3. Alshamrani, A.; Myneni, S.; Chowdhary, A.; Huang, D. A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 1851–1877. [CrossRef]
4. Yang, L.X.; Li, P.; Zhang, Y.; Yang, X.; Xiang, Y.; Zhou, W. Effective repair strategy against advanced persistent threat: A differential game approach. *IEEE Trans. Inf. Forensics Secur.* **2018**, *14*, 1713–1728. [CrossRef]
5. Li, P.; Yang, X.; Xiong, Q.; Wen, J.; Tang, Y.Y. Defending against the advanced persistent threat: An optimal control approach. In *Security and Communication Networks*; Hindawi: London, UK, 2018.
6. Cai, G.; Wang, B.; Hu, W.; Wang, T. Moving target defense: state of the art and characteristics. *Front. Inf. Technol. Electron. Eng.* **2016**, *17*, 1122–1153. [CrossRef]
7. Carvalho, M.; Ford, R. Moving-target defenses for computer networks. *IEEE Secur. Privacy* **2014**, *12*, 73–76. [CrossRef]
8. Okhravi, H.; Hobson, T.; Bigelow, D.; Streilein, W. Finding focus in the blur of moving-target techniques. *IEEE Secur. Privacy* **2013**, *12*, 16–26. [CrossRef]
9. Jajodia, S.; Gohsh, A.K.; Swarup, V.; Wang, C.; Wang, X.S. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*; Springer Science & Business Media: New York, NY, USA, 2011.
10. Zhou, Y.; Cheng, G.; Jiang, S.; Hu, Y.; Zhao, Y.; Chen, Z. A cost-effective shuffling method against DDoS attacks using Moving Target Defense. In Proceedings of the 6th ACM Workshop on Moving Target Defense, London, UK, 11 November 2019.
11. Wing, J.M.; Manadhata, P.K. *Measuring a System'S Attack Surface: Technical Report CMU-CS-04-102*; Carnegie Mellon University: Pittsburgh, PA, USA, 2004.
12. Woo, S.; Park, K.; Moon, D.; Kim, I. Moving Target Defense Research Trend Based on Network Address Mutation. *Rev. Kiisc* **2018**, *28*, 5–11.
13. Jafat Haadi, J.; Al-Shaer, E.; Duan, Q. Openflow random host mutation: transparent moving target defense using software defined networking. In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks, Helsinki, Finland, 13 August 2012.
14. Yue-Bin, L.; Wang, B.S.; Wang, X.F.; Hu, X.F.; Cai, G.L.; Sun, H. RPAH: Random port and address hopping for thwarting internal and external adversaries. *IEEE Trustcom/BigDataSE/ISPA* **2015**, *1*, 263–270.
15. Dunlop, M.; Groat, S.; Urbanski, W.; Marchany, R.; Tront, J. Mt6d: A moving target ipv6 defense. In Proceedings of the IEEE Military Communications Conference, Baltimore, MD, USA, 7–10 November 2011.
16. Zangeneh, V.; Shajari, M. A cost-sensitive move selection strategy for moving target defense. *Comput. Secur.* **2018**, *75*, 72–91. [CrossRef]
17. Lei, C.; Zhang, H.Q.; Tan, J.L.; Zhang, Y.C.; Liu, X.H. Moving target defense techniques: A survey. In *Security and Communication Networks*; Hindawi: London, UK, 2018.
18. Sun, J.; Sun, K. DESIR: Decoy-enhanced seamless IP randomization. In Proceedings of the IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, San Francisco, LA, USA, 10–15 April 2016.
19. Yuill, J.; Denning, D.; Feer, F. Using deception to hide things from hackers: Processes, principles, and techniques. *J. Inf. Warf.* **2006**, *5*, 26–40.

20. Haller, N.; Metz, C.; Nesser, P.; Straw, M. A One-Time Password System. Available online: https://tools.ietf.org/html/rfc2289 (accessed on 13 April 2020).
21. Erdem, E.; Sandıkkaya, M.T. OTPaaS—One time password as a service. *IEEE Trans. Inf. Forensics Secur.* **2019**, *14*, 743–756. [CrossRef]