Proceedings of the 20th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 145–154 ISSN 2300-5963 ACSIS, Vol. 43
DOI: 10.15439/2025F3136

Hybrid mutation MTD solution with dedicated SDN agents

Mateusz Fronczyk
Warsaw University of Technology
ul. Nowowiejska 15/19, 00-665 Warszawa, Poland
Email: mateusz.fronczyk.stud@pw.edu.pl

Mariusz Rawski
0000-0002-7489-0785
Warsaw University of Technology
ul. Nowowiejska 15/19, 00-665 Warszawa, Poland
Email: mariusz.rawski@pw.edu.pl

***Abstract*—Reconnaissance is a crucial stage of cyberattacks, enabling attackers to gather information about system vulnerabilities. In computer networks, data regarding addressing and transmission paths is especially sensitive. This paper introduces a concept for a hybrid mutation system based on Software Defined Networking (SDN), combining address and data path mutation to improve security. The system employs central network management and pseudorandom, temporary addressing, which is periodically reconfigured. In contrast to existing Moving Target Defense (MTD) address mutation methods, end devices connect through dedicated agents, which can be implemented using a SmartNIC. The agent modifies packet headers at a network edge to obfuscate address information, minimizing the processing burden on SDN switches inside a network. The objective is to hinder attackers from discovering network details that could be exploited. A prototype was implemented using typical SDN components and containerized end devices. Tests confirmed the system's correctness and effectiveness in protecting the network structure and communication paths. This approach enhances the confidentiality of network parameters and limits the information available to potential attackers, making reconnaissance significantly more difficult, while minimizing SDN network control overhead.**

I. Introduction

NETWORK security is critical in today's digital landscape, as it protects sensitive data from unauthorized access and cyber threats while ensuring the integrity and availability of network resources. The network attack cycle spans several steps. The Cyber Kill Chain illustrates this cycle, detailing the stages an attacker typically goes through to successfully breach a network, from reconnaissance and weaponization to delivery, exploitation, installation, command and control, and finally, actions on objectives [1].

The reconnaissance is the initial phase where attackers gather information about their target, identifying potential vulnerabilities and entry points that can be exploited to gain unauthorized access. Stopping an attacker at the reconnaissance phase of a cyber attack provides significant security advantages by preventing further exploitation and reducing the overall risk to a network. However, spotting reconnaissance activity can be challenging due to several factors that make it stealthy and difficult to differentiate from legitimate network traffic. Traditional computer networks are inherently static, making them predictable targets for attackers who exploit static IP addresses, unchanged software stacks, and consistent routing paths to plan and execute cyberattacks. This predictability enables adversaries to conduct prolonged reconnaissance, identify vulnerabilities, and launch precise, well-timed exploits with minimal resistance. The static nature of computer networks gives the attacker a significant advantage over the defender, as attackers can use stealthy techniques to gather information over a long time to identify weaknesses without raising alarms [2].

A proposed solution to this issue was introduced in [3] as part of the USA Federal Networking and Information Technology Research and Development program, which seeks innovative approaches to address critical cybersecurity challenges. This approach, known as Moving Target Defense (MTD), represents a fundamental shift in securing computer networks. The core principle of MTD is that an attack, if successful, can only be effective once, as the system is continuously reconfigured to prevent the reuse of the same attack vector. By dynamically altering the attack surface, MTD increases the complexity and cost for attackers, reduces the window of vulnerability exposure, and enhances overall system resilience. Moving Target Defense mitigates this risk by introducing continuous, randomized changes to network attributes such as IP addresses, system configurations, and routing paths. By dynamically altering the attack surface, MTD disrupts adversarial strategies, forcing attackers to constantly reassess their tactics, thereby increasing the cost and complexity of cyber intrusions [4]. Unlike traditional static defenses relying on perimeter security mechanisms, MTD adopts a proactive security posture that introduces unpredictability while ensuring system reliability for legitimate users.

Numerous MTD strategies have been proposed for securing computer networks [5]. One of the key principles of MTD in networks is IP Address Mutation, which involves frequently changing the IP addresses of critical assets, preventing attackers from accurately mapping the network [6][7]. This technique disrupts traditional reconnaissance methods and limits the effectiveness of targeted attacks. Another effective approach is Dynamic Network Topology, which emphasizes

The research was carried out on devices cofunded by the Warsaw University of Technology within the Excellence Initiative: Research University (IDUB) programme.

dynamic changes in network architecture to confound attackers [8]. Route Mutation and Traffic Obfuscation techniques further enhance network security by dynamically altering packet routes, making it difficult for attackers to eavesdrop or execute man-in-the-middle attacks. This method ensures that sensitive data does not travel predictable paths, increasing the difficulty of successful network intrusions [9]. Service and Port Randomization also prevents attackers from leveraging known vulnerabilities by frequently changing open ports and service access points [10].

The benefits of MTD for networks are substantial. It disrupts reconnaissance efforts, reduces the attack window, and forces adversaries to adapt continuously, increasing their resource expenditure and reducing their ability to execute long-term attacks. Moreover, MTD enhances cyber resilience by introducing self-adaptive security measures that dynamically respond to emerging threats. However, challenges such as performance overhead, compatibility with legacy systems, and the need for intelligent automation must be addressed to optimize MTD implementation.

Software-Defined Networking (SDN) technology plays a key role in implementing most Moving Target Defence strategies, enabling dynamic, automated, and adaptive networking mechanisms. This technology provides programmability, flexibility, and scalability, making it possible to rapidly adjust network configurations, which constitutes the core of most network-based MTD methods. However, these frequent and unpredictable changes introduce significant stress on SDN switches and controllers, impacting performance, stability, and resource utilization.

As noted in [11], while centralization in SDN improves control and flexibility, it also introduces new attack vectors. Offloading certain security functions to edge components can mitigate these risks and improve scalability. With the advent of Smart Network Interface Cards (SmartNICs), there is potential to alleviate some of these pressures by offloading certain processing tasks from the SDN controllers and switches, thereby enhancing overall system efficiency and enabling more robust MTD solutions that can adapt in real-time without compromising network integrity. SmartNICs represent a significant evolution in network interface technology, offering enhanced functionality that extends far beyond traditional NICs. SmartNICs integrate programmable processors, memory, and dedicated hardware accelerators at their core, enabling them to offload network processing tasks [12].

This paper presents a hybrid addressing and route mutation mechanism that utilizes dedicated host agents with a lightweight SDN switch system. These host agents are designed to be implemented using SmartNICs, offloading specific tasks from core SDN switches. This approach not only improves the scalability of network management but also enhances performance by allowing core switches to focus on high-level routing decisions while host agents handle lower-level address mutation tasks efficiently.

II. Related Works

Several solutions based on the concept of address mutation and dynamic routing in network environments built on SDN architecture can be found in scientific literature.

One example of applying address mutation in SDN networks is the OpenFlow Random Host Mutation (OF-RHM) mechanism [13]. Each host is assigned a virtual IP address used during network communication in this solution. Importantly, these addresses are periodically changed, significantly complicating attacks based on traffic monitoring (e.g., sniffing) or network topology analysis. The address mutation process is handled by the switches, which also manage data traffic. Although this approach improves transmission security, it increases the load on the switches, which can negatively impact the overall performance and scalability of the network.

Another approach was presented in [14], where a routing algorithm based on fake addressing was proposed. In this model, switches establish paths between hosts using randomly assigned addresses, while the SDN controller manages the mapping of temporary addresses to actual ones. Despite its security advantages, this solution involves several significant challenges. Firstly, the mutation process executed on the switches generates additional load for these devices. Secondly, the path selection may lead to temporary network congestion, especially in more complex topologies, due to excessive packet flooding during route updates.

In [15], the authors proposed using local NAT tables on each switch. In this model, a dedicated MTD unit responsible for generating temporary addressing is connected to each switch. As a result, the SDN controller is relieved of some responsibility – its role is limited to initializing data transmission paths. Although this reduces the controller's load, the switches still need to handle the address translation process, which remains challenging for their performance.

On the other hand, the solution presented in [16] introduces an additional layer of switches called agents directly connected to end devices. These agents take over the task of address mutation, thereby offloading the main switches responsible for packet routing in the network. This approach limits the impact of mutation on the core infrastructure; however, it introduces another challenge – temporary addressing is created for each individual connection, which may lead to significant load on the SDN controller, especially in high-traffic environments.

III. MTD Concept

This paper proposes a solution based on the Moving Target Defense technique, utilizing a hybrid mutation approach. The proposed method introduces dynamic mutations at both the data transmission path level and the addressing level, specifically targeting the second and third layers of the OSI model, namely the MAC and IP address layers.

Our solution leverages the flexibility and centralized control offered by Software-Defined Networking, where the SDN controller plays a pivotal role. The controller is tasked with orchestrating both path and address mutations and dynamically installing flow rules on SDN switches throughout the network

via the OpenFlow protocol. To support the address mutation process, we introduce lightweight, specialized agents in the form of thin SDN switches that are designed to manipulate packet headers in real time.

These agents are deployed at the network edge, typically on end devices or SmartNICs. They are responsible for executing packet-level transformations in accordance with mutation policies received from the SDN controller. Each agent is capable of performing real-time mutation of MAC and IP addresses. The agents operate transparently to the applications running on the host devices, ensuring seamless communication while continuously altering address mappings to thwart reconnaissance and tracking attempts by potential adversaries.

By offloading the mutation tasks from the core SDN switches to these edge agents, the architecture reduces the processing overhead on network infrastructure components, improves scalability, and enables fine-grained control over address mutation. The result is a dynamic, resilient, and secure communication environment that significantly raises the bar for attackers attempting to exploit static network configurations.

The overall architecture of the proposed system is illustrated in Fig. 1, highlighting the interplay between the SDN controller, the network switches, and the deployed agents.

To establish a connection between devices, the controller performs a pseudo-random selection of one of the available paths labeled 1, 2, and 3 in the diagram. Data exchange takes place over the selected route for the duration defined by the mutation interval.

During operation, packets transmitted in the network are subject to modification to obscure their actual source and destination addresses. Temporary false addressing is applied, referred to as $tMAC$ (temporary MAC address) and tIP (temporary IP address). After the mutation interval expires, the controller generates new temporary addresses and selects new transmission paths for each end device connected to the network.

The real addresses, corresponding to the physical configuration of device interfaces, are denoted as $pMAC$ (permanent MAC address) and pIP (permanent IP address). From the perspective of communicating hosts, it is impossible to discover the actual address of the machine involved in data exchange, ensuring complete address obfuscation within the system. As a result, all packets transmitted within the network contain only temporary, falsified source and destination addresses in their headers.

This mechanism can operate successfully provided all communicating devices, including SDN switches, remain under the organization's control. The system also assumes that the organization has access to a large address pool within a given subnet, allowing for effective masking of real IP addresses. To avoid raising suspicion from a potential attacker, temporary IP addresses are selected from the available address space within the corresponding subnet.

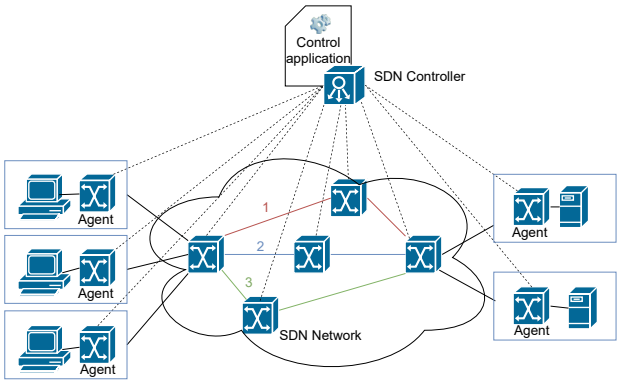

Fig. 1. The architecture of the proposed network

A. Connection Establishment and Handling

Fig. 2 presents a sequence diagram illustrating the phases of a sample connection establishment between *Host 1* and *Server 1*. This process includes:

1) **ARP Request for the *DNS Server***
   Communication with the *DNS Server* is the first step in establishing the connection. Assuming that the ARP tables on the end devices are initially empty, *Host 1* sends an ARP request to the broadcast address (ff:ff:ff:ff:ff:ff) asking for the MAC address of the *DNS Server*. The *H1 Agent* intercepts this request and replies to *Host 1* with a packet containing a temporary MAC address for the *DNS Server*, marked as *tMACDNS* in the diagram.
2) **DNS Query**
   Host 1, now knowing the temporary MAC address of the *DNS Server* (*tMACDNS*), sends a query requesting the IP address of a specific domain. In the presented diagram, the DNS query concerns the domain *server1.pl*, corresponding to the device *Server 1*. The query is received by the *H1 Agent* and then sent to the *SDN Controller* using a `PACKET_IN` message.
   Upon receiving the query, the *SDN Controller* performs a pseudo-random path selection to determine through which switch the communication will proceed. It also verifies whether the requested domain is handled by the *DNS Server*. If the domain does not exist, the connection attempt fails. If the domain is valid, the controller uses temporary addresses created for the current mutation state.
   One of these temporary addresses is the fake IP address of *Server 1* (*tIPS1*). The *SDN Controller* uses this address to modify the DNS server's configuration file, which maps actual server addresses to domain names.
   In the next step, the controller uses the temporary addressing to generate rules for appropriate agents and switches in the network. These rules are installed using `FLOW_MOD` messages. Rules are installed on the *H1*, *S1*, and *DNS* agents. The rules for *H1* and *S1* handle address obfuscation and ARP query handling

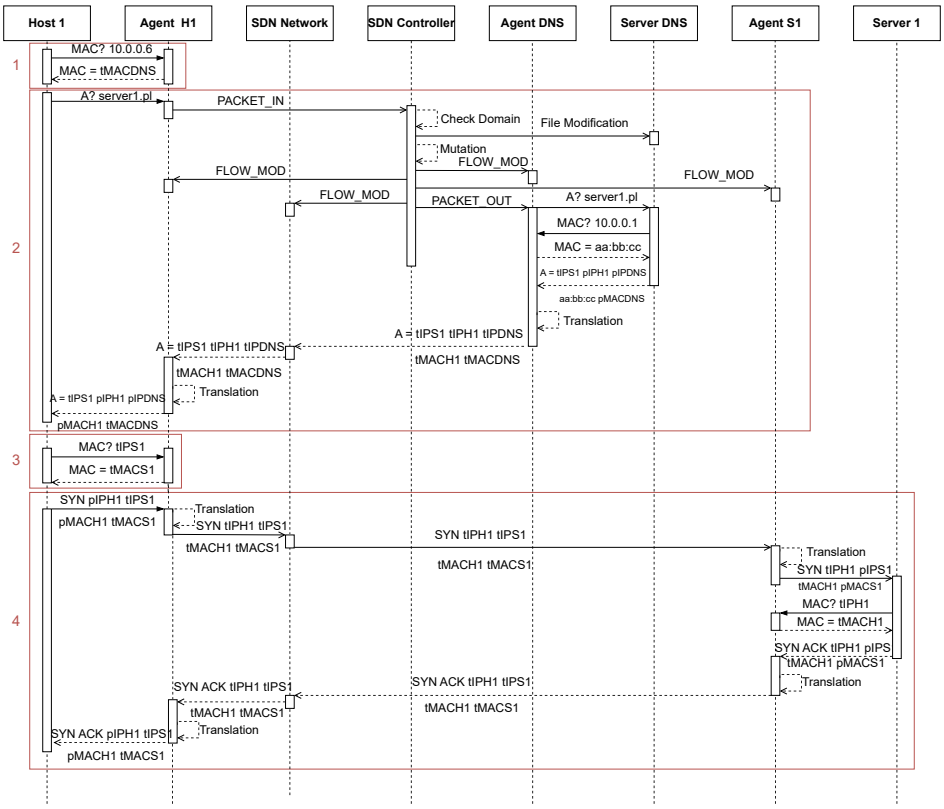

Fig. 2. Connection Establishment and Handling Diagram

for temporary IP addresses. The rule installed on the *DNS Agent* ensures that the DNS response is delivered to *Host 1* through the network. To achieve this, the *SDN Controller* additionally installs rules on *Host 1*'s agent and on the switches in the *SDN Network* to allow proper delivery of the response from the *DNS Server*. The rules installed on the *SDN Network* switches serve exclusively to forward packets between end devices using obfuscated addressing.

The DNS query is directly forwarded to the *DNS Agent* and then to the *DNS Server*. To send a response, the *DNS Server* must obtain the MAC address of *Host 1*, from which the original query was received. It sends an ARP request, which is intercepted by the *DNS Agent*, who replies with a fake MAC address `00:00:00:aa:bb:cc`. Due to the complexity of the proposed system's logic, the *DNS Agent* is configured with a flow rule that returns this MAC address for any IP in response to an ARP request. After receiving the ARP response, the *DNS Server* sends back a DNS response containing the fake IP address corresponding to the requested domain (*tIPS1*). The *DNS Agent*, upon receiving this response, sends it to the *SDN Controller*, which then forwards the packet to the *H1 Agent* via the *SDN Network*. While sending the response into the network, the *DNS Agent* performs address translation, inserting temporary addressing into the packet header: *tMACDNS* and *tIPDNS* for the source, and *tMACH1* and *tIPH1* for the destination. When the *H1 Agent* receives the response, it performs reverse translation by replacing the destination addresses with the real addressing of *Host 1* (*pMACH1* and *pIPH1*).

3) **ARP Request for *Server 1***
After receiving the DNS response containing *tIPS1*, *Host 1* sends an ARP request to obtain the MAC address assigned to this IP address. The rule installed by the *SDN Controller* on the *H1 Agent* handles the request, returning the temporary MAC address of *Server 1*, marked in the diagram as *tMACS1*.

4) **Communication Between Host and Server**
The next part of the diagram presents a sample TCP connection established between *Host 1* and *Server 1*. In the first phase, *Host 1* sends a SYN message as the initial step of the TCP three-way handshake. The packet header includes the real source address (*pIPH1* and *pMACH1*) and the temporary destination address (*tIPS1* and *tMACS1*). Upon receiving it, the *H1 Agent* performs address translation using *tMACH1* and *tIPH1*. The packet sent into the network has fully obfuscated addressing until it reaches the *S1 Agent*, who then

performs reverse translation by replacing the temporary destination address (*tMACS1* and *tIPS1*) with the real address (*pMACS1* and *pIPS1*). After *Server 1* successfully receives the segment, it replies with a SYN-ACK. Meanwhile, the *S1 Agent* handles an ARP query for the *Host 1* address mapping, returning the temporary address *tMACH1*. Since communication is bidirectional, *S1 Agent* once again performs address translation for the response, and on the receiving side, the *H1 Agent* performs reverse translation.

B. Mutation State Change During Connection Duration

One of the assumptions of the system operation is the continuation of the connection when a mutation state change occurs. Fig. 3 illustrates the sequence diagram that fulfills this assumption. The diagram shows a fragment of communication between *Server 1* and *Host 1*, which is downloading a file from the server. It is assumed that the file is large enough for the download process to be time-consuming. It is also assumed that the mutation interval *Tm* is short enough to change a mutation state during the file download process.

After the connection is successfully established, the file download from *Server 1* begins. Communication between *Host 1* and *Server 1* takes place using segments containing the $PUSHACK$ and ACK flags. The segment sent by *Server 1* with the $PUSHACK$ flag includes the data segment of the file requested by *Host 1*. The ACK segment, on the other hand, is the response from *Host 1* indicating successful reception of the segment from *Server 1*. This communication continues until *Server 1* sends all segments. At this point, *Server 1* sends a FIN flag segment, informing *Host 1* that all segments have been transmitted and initiating the connection termination process.

During the connection establishment, the *SDN Controller* places flow rules in the switch flow tables, enabling communication using temporary addressing marked in the diagram as addresses *t0*. These rules have a defined time *Tm*, after which they are permanently removed from the switch. When the rule responsible for address obfuscation on the *Agent H1* is removed, it sends a FLOW_REMOVED message to the *SDN Controller*, informing it that the rule has been deleted. The *SDN Controller* then re-selects the path for communication. Next, using the newly generated addressing in state *t1*, it installs flow rules on the switches responsible, in the case of *Agent H1* and *Agent S1*, for translating addresses *t1*, and for switches in the *SDN Network*, for communication using the *t1* addresses along the new path. After the rules are installed, the client-server communication continues with the new data path state and obfuscated addressing.

IV. Implementation

The experimental environment for the designed system was implemented using the Containernet network emulator, running on a virtual machine with Ubuntu 20.04. Open vSwitch (OVS) was used to implement the agents and SDN switches. These components form the foundation of a software-defined network architecture, which supports advanced dynamic behavior and fine-grained traffic control.

The SDN network is managed by the Ryu controller, which runs a dedicated control application written in Python. This application serves as the central intelligence of the system, orchestrating all major network operations such as address mutation, communication path management and DNS response manipulation. It interacts continuously with switches and agents, using the OpenFlow protocol to install and update flow rules that define how packets are forwarded throughout the network.

One of the core features of the control application is its dynamic addressing mechanism. At fixed intervals, the system triggers a mutation process that generates random temporary IP and MAC addresses for all endpoints in the network, including hosts, servers, and the DNS server. These addresses replace the real ones in communication flows, significantly reducing the feasibility of reconnaissance and tracking by potential attackers. The mutation runs asynchronously in the background, ensuring uninterrupted operation of other system components.

In addition to address mutation, the control application configures agent-hosted rules to intercept and respond to ARP queries. When hosts send ARP requests for the DNS server, the agents reply with dynamically assigned fake MAC addresses.

Another essential component of the architecture is a DNS server built using the Python $Dnslib$ library. It processes traditional DNS queries but is tightly integrated with the control logic. Upon receiving a DNS request, the control application extracts the queried domain, verifies it against a known mapping, and assigns a temporary IP address corresponding to the correct service endpoint. The DNS server configuration is then updated in real time to reflect the new address, ensuring all clients receive the current valid mapping.

Once a DNS query is resolved, the control application establishes the necessary flow rules to guide the DNS response and subsequent communication across the SDN switches and agents. For every transmission, source and destination addresses are translated to temporary counterparts, and then restored on the receiving end, enabling seamless and transparent bidirectional communication.

Communication paths themselves are also subject to mutation. The system periodically selects different routes for data transfer from a predefined set of options. New flow rules are installed along the newly chosen path, while outdated ones are removed.

All flow rules are configured with hard timeouts to ensure that temporary addressing remains effective. The control application is notified when a rule expires and installs a new rule set with updated addresses and paths. This dynamic rule replacement ensures continued communication fidelity while preserving the ephemeral nature of all addressing and routing information.

The overall logic of the control application described above is presented in Algorithm 1.

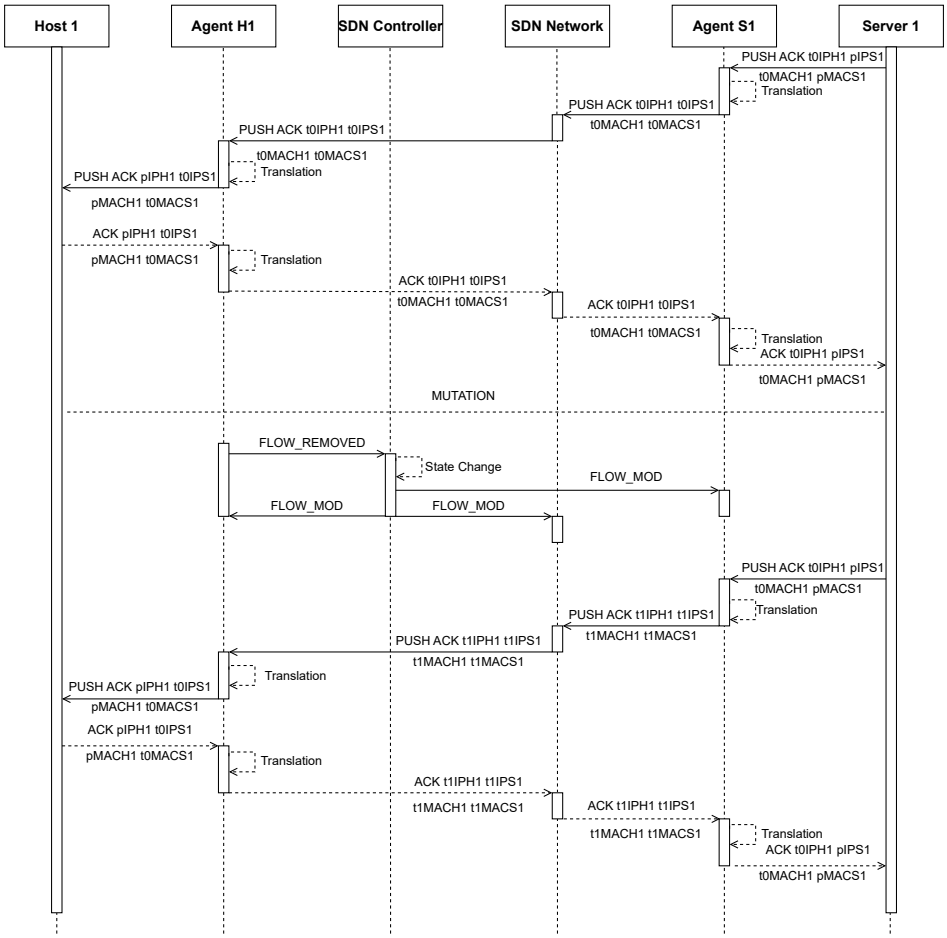

Fig. 3. Mutation State Change During the Connection

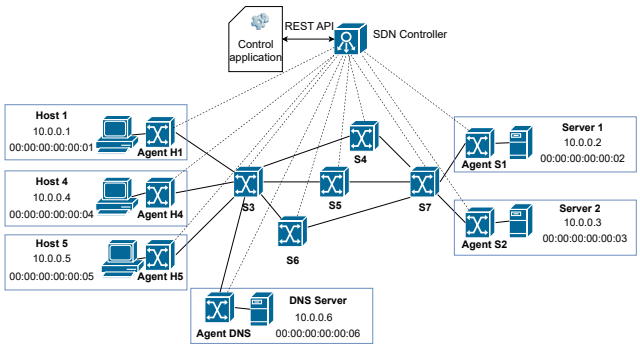

Fig. 4. SDN Network Topology in the Research Environment

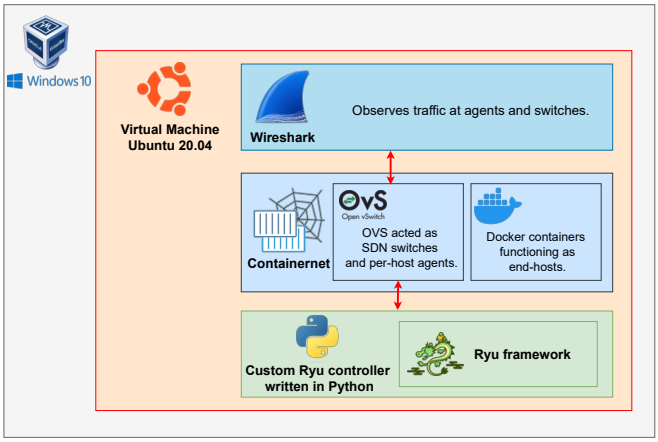

Fig. 5. Logical architecture of the implemented environment

The structure of the implemented network, including address allocation for individual components, is illustrated in Fig. 4, and the logical system architecture is shown in Fig. 5.

V. Validation and Results

A series of test scenarios were developed to verify the implemented system's operation.

Algorithm 1 Control Application Logic for SDN Mutation System

Require: Active SDN controller, initialized switches, known domain-to-server mappings
Ensure: Continuous operation with dynamic addressing and path mutation

Initialize mappings and path set
Start background **MutationLoop**
OnSwitchFeaturesReceived
if host agent **then**
    Install ARP rule, redirect DNS to controller
else if DNS agent **then**
    Install ARP rule
end if
OnPacketIn
if DNS query for known domain **then**
    Select endpoint, assign temporary IP/MACs
    Store mappings, update DNS config
    Install translation and ARP rules
    Select path, install forwarding rules
    Forward modified DNS packet
end if
OnFlowRemoved
if ARP rule expired **then**
    Reinstall ARP rule
else if IP flow expired **then**
    Identify flow, retrieve temp addresses
    Select path, reinstall updated rules
end if
MutationLoop (background)
while true **do**
    Sleep for interval, generate temp IP/MACs
    Update mappings, reinstall rules
end while

A. Address Mutation

The first test scenario illustrates network traffic on various devices belonging to the selected path during communication between *H4* and *Server 2* using the $ping$ command.

Fig. 8 shows traffic on the interface of the agent connected to *H4*. Initially, the agent responds to an ARP query regarding the DNS server, returning the temporary MAC address (*t_MAC_DNS*). Then, the host sends a DNS query for the IP address of the server domain $server2.pl$. After forwarding the query to the controller, it is sent to the DNS server, which returns a response containing the temporary IP address of the target server (*t_IP_S2*).

To begin the transmission of ICMP packets, the host must still obtain the MAC address assigned to the server. In response to the ARP query, the agent returns the temporary MAC address of the server (*t_MAC_S2*). Following this, correct communication occurs, using the temporary destination addressing.

The source addressing is obscured when the packet passes through the agent. The agent then replaces the real addressing with the temporary one corresponding to the given host (*t_MAC_H4* and *t_IP_H4*). Fig. 9 shows the observation of traffic on a selected switch in the *SDN Network* located on the communication path. The ICMP packets contain both temporary source and destination addressing.

The observation of traffic on the *Agent S2* device port, connecting the agent to the server, is shown in Figure 10. Upon receiving the appropriate packet from the network, the agent performs a reverse translation, replacing the temporary server addressing with the real one, labeled *p_MAC_S2* and *p_IP_S2*, allowing the response to be sent to the host.

B. Change of Address Mutation State

As part of the address mutation mechanism, after the time *Tm* has passed, communication in the system takes place using the temporary addressing of the new state. In the following Fig. 6 and Fig. 7, the communication between *Host 1* and *Server 1* during the download of an HTTP page is presented, using temporary addresses in two different states of the system.

During the first two connections (lines 1-27 and 3-6 of listings in Fig. 6 and Fig. 7 respectively), *Host 1* communicates with *Server 1* using the temporary address 10.0.0.165, and the server 10.0.0.53. After the time *Tm*, defined in the test as 10 seconds, the subsequent page download (lines 29-41 and 7-8) takes place using a new pair of temporary addresses: 10.0.0.237 for *Host 1* and 10.0.0.102 for *Server 1*.

```
1  root@h1:/# wget http://server1.pl:8080
2  --2025-01-07 17:28:49--  http://server1.pl:8080/
3  Resolving server1.pl (server1.pl)... 10.0.0.53
4  Connecting to server1.pl (server1.pl)
5  |10.0.0.53|:8080... connected.
6  HTTP request sent, awaiting response... 200 OK
7  Length: 1028 (1.0K) [text/html]
8  Saving to: 'index.html'
9
10 100%[============>] 1,028       --.-K/s   in 0s
11
12 2025-01-07 17:28:49 (244 MB/s) -
13 'index.html' saved [1028/1028]
14
15 root@h1:/# wget http://server1.pl:8080
16 --2025-01-07 12:28:52--  http://server1.pl:8080/
17 Resolving server1.pl (server1.pl)... 10.0.0.53
18 Connecting to server1.pl (server1.pl)
19 |10.0.0.53|:8080... connected.
20 HTTP request sent, awaiting response... 200 OK
21 Length: 1028 (1.0K) [text/html]
22 Saving to: 'index.html.1'
23
24 100%[============>] 1,028       --.-K/s   in 0s
25
26 2025-01-07 17:28:52 (213 MB/s) -
27 'index.html.1' saved [1028/1028]
28
29 root@h1:/# wget http://server1.pl:8080
30 --2025-01-07 17:29:09--  http://server1.pl:8080/
31 Resolving server1.pl (server1.pl)... 10.0.0.102
32 Connecting to server1.pl (server1.pl)
33 |10.0.0.102|:8080... connected.
34 HTTP request sent, awaiting response... 200 OK
35 Length: 1028 (1.0K) [text/html]
36 Saving to: 'index.html.2'
37
38 100%[============>] 1,028       --.-K/s   in 0s
39
40 2025-01-07 17:29:09 (213 MB/s) -
41 'index.html.2' saved [1028/1028]
```

Fig. 6. Downloading a page from *Server 1* on device *Host 1*

```
root@server1:/# python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 ...
10.0.0.165 - - [07/Jan/2025 17:28:49]
"GET / HTTP/1.1" 200 -
10.0.0.165 - - [07/Jan/2025 17:28:52]
"GET / HTTP/1.1" 200 -
10.0.0.237 - - [07/Jan/2025 17:29:09]
"GET / HTTP/1.1" 200 -
```

Fig. 7. Received requests of *Server 1*

C. Mutation of Path and Address State During the Connection

The implemented system supports mutation during the connection, ensuring continuous communication. In the test on device *Host 5*, a 1GB file was downloaded from *Server 1*. The time *Tm* was set to 5 seconds, allowing for the observation of frequent mutation state changes in the system.

Upon establishing a connection with *Server 1*, *Host 1* received the following messages, shown in Fig. 11:

- temporary MAC address of the *DNS Server* 23:DA:A9:89:27:51 (*t_MAC_DNS*),
- temporary IP address of *Server 1* 10.0.0.185 (*t_IP_S1_t0*),
- temporary MAC address of *Server 1* B1:56:ED:C4:2D:06 (*t_MAC_S1_t0*).

Subsequent communication involves downloading the file "plik.bin" from *Server 1*. Fig. 12 shows the traffic observed on switch *S5* (chosen for path mutation) with completely obfuscated addressing. *Host 5* uses the temporary address D1:58:C0:F7:AE:3C (*t_MAC_H5_t0*) and 10.0.0.247 (*t_IP_H5_t0*). Fig. 13 depicts the situation of a mutation state change. After the time *Tm* has passed, the flows on the corresponding switches are updated, enabling continued communication with a new address and path. In the new mutation state for the path, the switch changes to *S4*, through which the segments are transmitted. Additionally, new temporary addresses appear in the packet headers:

- MAC address of *Host 5* 93:97:15:01:69:DC (*t_MAC_H5_t1*),
- IP address of *Host 5* 10.0.0.236 (*t_IP_H5_t1*),
- MAC address of *Server 1* D3:03:0A:06:8F:8A (*t_MAC_S1_t1*),
- IP address of *Server 1* 10.0.0.21 (*t_IP_S1_t1*).

Upon the mutation state change, retransmission of the segments occurs. This happens so quickly that the connection continues with the new addressing shortly afterward, enabling the entire file to be downloaded.

VI. Discussion

The test scenarios described above validate the correct functioning of the proposed Moving Target Defense system based on hybrid mutations at both the MAC and IP layers, as well as dynamic path adjustments during communication.

The results from the Address Mutation test confirm that the agents correctly perform address obfuscation and translation at the network edges. Temporary addresses are assigned and managed without impacting the continuity of communication from the hosts' perspective. This behavior is evident from the consistent and uninterrupted exchange of ICMP packets and successful DNS resolution processes involving mutated addressing.

The Change of Address Mutation State scenario demonstrates the system's ability to transition between different temporary address states during normal operations seamlessly. After the predefined timeout interval (*Tm*), new sets of temporary IP addresses are allocated and applied transparently, while ongoing services (such as HTTP downloads) continue unaffected. This dynamic reassignment capability significantly increases the complexity for any adversary attempting to map the network or track devices, reinforcing the defensive posture.

Finally, the Mutation of Path and Address State During the Connection scenario showcases the most demanding feature of the system—live mutation of both the data path and addressing during an active session. During a large file transfer, the system's behavior proves that the agents and controller can coordinate effectively to update switch flows and reassign addressing without disrupting the data stream. Segment retransmissions and address switching occur within such short intervals that the host application layer remains unaware of the underlying changes.

The tests and observed results prove that the proposed solution functions as intended. It achieves its core objective of creating a dynamic and unpredictable network communication environment, effectively increasing potential attackers' difficulty in exploiting static addressing schemes or consistent routing paths.

Although effective in the test environment, the system uses temporary IPv4 addresses from a limited subnet. In larger networks, this may lead to address exhaustion. Supporting IPv6 would expand the address pool and enhance scalability and obfuscation.

VII. Conclusion

This paper presents an MTD system employing a hybrid mutation strategy involving address and path alterations to enhance network security. By cyclically mutating these elements during connection initiation and throughout the session duration, the proposed approach significantly increases the complexity of network reconnaissance for potential attackers. The design leverages dedicated packet processing agents, which can be efficiently implemented on SmartNICs at end hosts, enabling high-performance execution with minimal impact on system resources.

The prototype implementation, tested in an SDN-based environment utilizing OVS switches, demonstrated that the mutation mechanism operates seamlessly without disrupting normal network operations. Experimental results validated the system's effectiveness in improving the security posture of corporate networks by dynamically and unpredictably altering attack surfaces. Future work will optimize mutation intervals, integrate adaptive intelligence for mutation decisions, and evaluate performance in larger and more diverse network topologies.

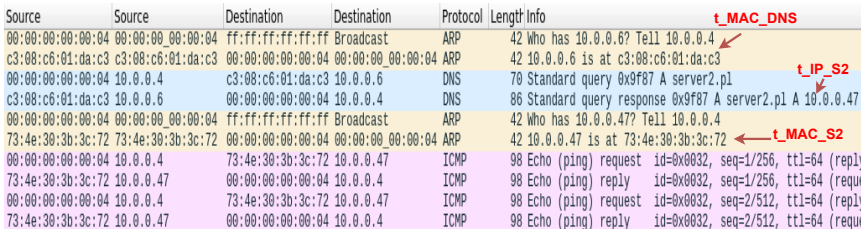

Fig. 8. Traffic observed on the port of the *Agent H4* device connecting the agent to the host

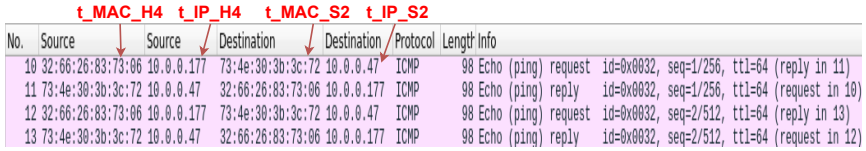

Fig. 9. Traffic observed on the switch in the *SDN Network*

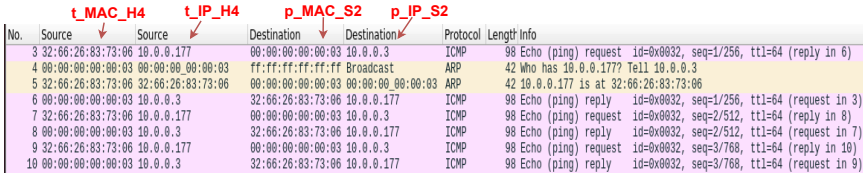

Fig. 10. Traffic observed on the port of the *Agent S2* device connected to the server

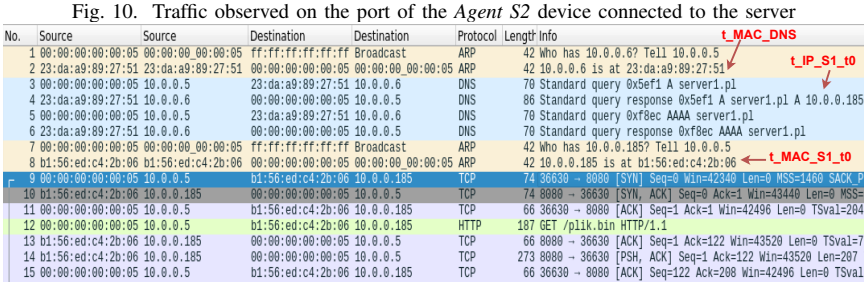

Fig. 11. Traffic observed on the *Agent H5* device

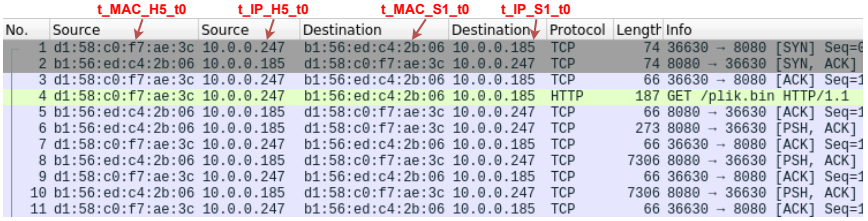

Fig. 12. The traffic observed on switch *S5* in state *t0*

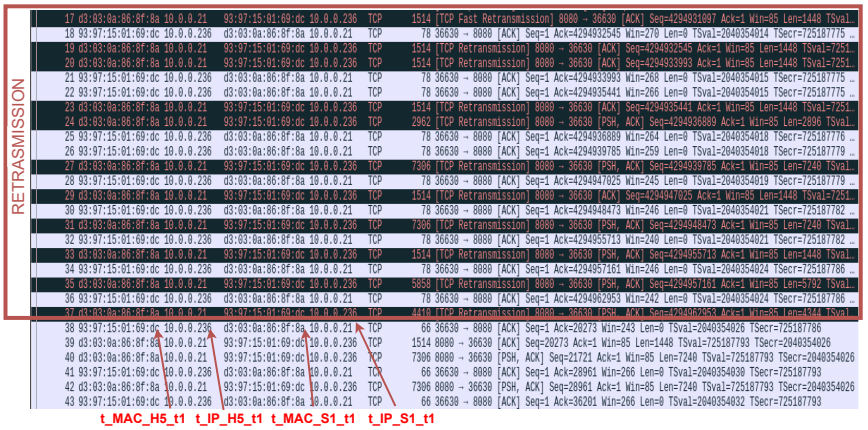

Fig. 13. Traffic observed on switch *S4* in state *t1*

REFERENCES

[1] M. S. Khan, S. Siddiqui, and K. Ferens, "A cognitive and concurrent cyber kill chain model," *Computer and Network Security Essentials*, pp. 585–602, 8 2017. doi: 10.1007/978-3-319-58424-9_34/FIGURES/4

[2] M. Albanese, "From cyber situational awareness to adaptive cyber defense: Leveling the cyber playing field," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11170 LNCS, pp. 1–23, 2018. doi: 10.1007/978-3-030-04834-1_1/FIGURES/12

[3] Fred, R. Lee, A. Acquisti, W. Horne, C. Palmer, A. Ghosh, D. Pendarakis, W. Sanders, E. Fleischman, H. Teufel III, and others Chong, "National cyber leap year summit 2009 co-chairs' report, networking and information technology research and development," 9 2009.

[4] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, 1st ed. Springer Publishing Company, Incorporated, 2011. ISBN 1461409764

[5] L. Jalowski, M. Zmuda, and M. Rawski, "A survey on moving target defense for networks: A practical view," *Electronics 2022, Vol. 11, Page 2886*, vol. 11, p. 2886, 9 2022. doi: 10.3390/ELECTRONICS11182886

[6] E. Al-Shaer, Q. Duan, and J. H. Jafarian, "Random host mutation for moving target defense," *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, vol. 106 LNICS, pp. 310–327, 2013. doi: 10.1007/978-3-642-36883-7_19

[7] C. Gudla and A. H. Sung, "Moving target defense discrete host address mutation and analysis in sdn," *Proceedings - 2020 International Conference on Computational Science and Computational Intelligence, CSCI 2020*, pp. 55–61, 12 2020. doi: 10.1109/CSCI51800.2020.00017

[8] M. Rawski, "Network topology mutation as moving target defense for corporate networks," *International Journal of Electronics and Telecommunications*, vol. 65, pp. 571–577, 2019. doi: 10.24425/IJET.2019.129814

[9] L. Zhang, Q. Wei, K. Gu, and H. Yuwen, "Path hopping based sdn network defense technology," *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery, ICNC-FSKD 2016*, pp. 2058–2063, 10 2016. doi: 10.1109/FSKD.2016.7603498

[10] A. Chowdhary, D. Huang, A. Alshamrani, and H. Liang, "Mtd analysis and evaluation framework in software defined network (mason)," *SDN-NFVSec 2018 - Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks and Network Function Virtualization, Co-located with CODASPY 2018*, vol. 2018-January, pp. 43–48, 3 2018. doi: 10.1145/3180465.3180473

[11] K. Cabaj, J. Wytrębowicz, S. Kukliński, P. Radziszewski, and K. T. Dinh, "SDN Architecture Impact on Network Security," in *Position Papers of the 2014 Federated Conference on Computer Science and Information Systems (FedCSIS)*, ser. Annals of Computer Science and Information Systems, M. Ganzha, L. Maciaszek, and M. Paprzycki, Eds., vol. 3. IEEE/Polish Information Processing Society, 2014. doi: 10.15439/2014F473 pp. 143–148. [Online]. Available: http://dx.doi.org/10.15439/2014F473

[12] E. F. Kfoury, S. Choueiri, A. Mazloum, A. Alsabeh, J. Gomez, and J. Crichigno, "A comprehensive survey on smartnics: Architectures, development models, applications, and research directions," *IEEE Access*, vol. 12, pp. 107 297–107 336, 2024. doi: 10.1109/ACCESS.2024.3437203

[13] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: transparent moving target defense using software defined networking," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: Association for Computing Machinery, 2012. doi: 10.1145/2342441.2342467. ISBN 9781450314770 p. 127–132. [Online]. Available: https://doi.org/10.1145/2342441.2342467

[14] S. Wang, L. Zhang, and C. Tang, "A new dynamic address solution for moving target defense," in *2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference*, 2016. doi: 10.1109/ITNEC.2016.7560545 pp. 1149–1152.

[15] A. Almohaimeed and A. Asaduzzaman, "A novel moving target defense technique to secure communication links in software-defined networks," in *2019 Fifth Conference on Mobile and Secure Services (MobiSecServ)*, 2019. doi: 10.1109/MOBISECSERV.2019.8686530 pp. 1–4.

[16] D. C. MacFarland and C. A. Shue, "The sdn shuffle: Creating a moving-target defense using host-based software-defined networking," in *Proceedings of the Second ACM Workshop on Moving Target Defense*, ser. MTD '15. New York, NY, USA: Association for Computing Machinery, 2015. doi: 10.1145/2808475.2808485. ISBN 9781450338233 p. 37–41. [Online]. Available: https://doi.org/10.1145/2808475.2808485