

MTDSense: AI-Based Fingerprinting of Moving Target Defense Techniques in Software-Defined Networking

Tina Moghaddam, Guowei Yang, Chandra Thapa, Seyit Camtepe, and Dan Dongseong Kim

Abstract—Moving target defenses (MTD) are proactive security techniques that enhance network security by confusing the attacker and limiting their attack window. MTDs have been shown to have significant benefits when evaluated against traditional network attacks, most of which are automated and untargeted. However, little has been done to address an attacker who is aware the network uses an MTD. In this work, we propose a novel approach named MTDSense, which can determine when the MTD has been triggered using the footprints the MTD operation leaves in the network traffic. MTDSense uses unsupervised clustering to identify traffic following an MTD trigger and extract the MTD interval. An attacker can use this information to maximize their attack window and tailor their attacks, which has been shown to significantly reduce the effectiveness of MTD. Through analyzing the attacker’s approach, we propose and evaluate two new MTD update algorithms that aim to reduce the information leaked into the network by the MTD. We present an extensive experimental evaluation by creating, to our knowledge, the first dataset of the operation of an IP-shuffling MTD in a software-defined network. Our work reveals that despite previous results showing the effectiveness of MTD as a defense, traditional implementations of MTD are highly susceptible to a targeted attacker.

Index Terms—Artificial intelligence, cyber attacks, intelligent cyber attacks, moving target defense, software-defined networking

I. INTRODUCTION

AS the world grows increasingly digital and more of the everyday workings of life and business come to rely on computer networks, securing network infrastructure is an ever-present problem. In this landscape, traditional networks resemble sitting ducks, as their static nature gives attackers an advantage [1]. More modern virtualized networks still underutilize their potential to respond to adaptive attacks by adopting traditional modes of operation despite their flexibility. Moving Target Defense (MTD) is a class of defense strategies that allow the defender to be more proactive [2]. In adopting MTD, the defender changes some properties of the attack surface to hinder the attacker. Proposed MTDs have changed

the network hosts’ IP addresses, transmission routes, and available applications to invalidate the attacker’s understanding of the network, reduce their windows of opportunity, and confuse them in their approach. Recently, MTDs are also increasingly being offered as consumer security solutions through companies such as SCIT Labs [3] and NexiTech [4]. As the defense and commercial usage of MTD increases, evaluating their true security against sophisticated attackers is increasingly important.

Since their introduction, MTDs have been shown to be effective in reducing the attacker’s chances of success [1], [2]. However, most evaluations assume that the attacker does not consider the MTD itself or does not even know it is being used as a defense mechanism. Previous work has mostly evaluated MTD against simple automated attackers, such as scanning worms and DDoS attacks [2], [5]. Very few works consider AI-powered attackers who can adjust their behavior given the MTD. Among these, skilled human attackers have been considered [6], as well as deanonymizing devices employing MAC address randomization [7], [8]. To the best of our knowledge, no work directly targets network MTD itself.

This paper addresses the gap by exploring how an attacker could overcome the MTD by targeting it directly in reconnaissance. Specifically, we look in detail into how the attacker can determine when an IP shuffling MTD is triggered in a software-defined network (SDN). Machine learning techniques are making it much simpler to find meaningful information among large volumes of noisy data at near-real-time speeds. We leverage this in analyzing passively collected network traffic between legitimate clients and servers. We propose a novel approach called MTDSense, which uses unsupervised clustering techniques to determine the absolute MTD trigger time and the duration between triggers, which we call the MTD interval. With this information, an attacker can plan the timing of their attacks, which has been shown to improve the attack success rate across all phases of a cyber attack by 40% [9]. Our work reveals that current MTD mechanisms are susceptible to attackers who target them during reconnaissance, even though there has been considerable work to show that they are effective against common known attacks.

Next, we investigate the fingerprint the MTD leaves in the network that allows the MTD trigger to be detectable using machine learning. By finding the unique timing features that reveal the operation of the MTD, we propose alternative update mechanisms to reduce the MTD fingerprint. Typical mechanisms for IP shuffling MTD are modeled on common

T. Moghaddam is with the School of Electrical Engineering and Computer Science, University of Queensland, St Lucia, QLD 4072, Australia, and also with Data61, Commonwealth Scientific and Industrial Research Organization, Sydney, NSW 2122, Australia (e-mail: t.moghaddam@uq.edu.au).

C. Thapa and S. Camtepe are with Data61, Commonwealth Scientific and Industrial Research Organization, Sydney, NSW 2122, Australia (e-mail: chandra.thapa@data61.csiro.au; seyit.camtepe@data61.csiro.au).

G. Yang and D.D. Kim are with the School of Electrical Engineering and Computer Science, University of Queensland, St Lucia, QLD 4072, Australia (e-mail: guowei.yang@uq.edu.au; dan.kim@uq.edu.au).

(Corresponding author: T. Moghaddam)

routing protocols of the internet and do not consider information leakage. Our novel update mechanisms deliberately time the update operations to minimize the information leak and make it harder for MTDSense to detect their operation. We experimentally evaluate MTDSense with respect to network and traffic size and MTD update mechanism using a real SDN testbed as well as supporting simulations.

The main contributions of this work are summarized as follows:

- We demonstrate that an Artificial intelligence (AI)-capable attacker can determine the timing of an IP shuffling MTD. To the best of our knowledge, this is the first work where network reconnaissance targets the MTD directly.
- We propose, implement, and evaluate two novel MTD update algorithms with the aim of reducing the MTD fingerprint. The susceptibility of these algorithms against the attack and the respective trade-offs are analyzed.
- We collect a dataset from an SDN using an IP shuffling MTD for defense covering both a simulated and testbed environment. The dataset varies a series of parameters of network and traffic size and comprises over 150 days of traffic data.

II. MTD IN SOFTWARE DEFINED NETWORKING

While MTD techniques have been studied in other systems, software-defined networking (SDN) introduced a new set of MTD techniques for large networks and accelerated research with the development of the OpenFlow protocol standard [2]. SDN decouples the control of the network from the physical devices. Decision making in SDNs is centralized to the SDN controller. In turn, SDN-enabled switches store network traffic rules in their flow tables, which can be programmed by the controller. This makes it possible to change system configurations on the fly and more frequently than in traditional decentralized systems, making implementing MTDs in SDNs much easier in comparison. As a result, a large portion of MTDs are implemented in SDNs [1].

Network level and specifically network address randomization MTD techniques are amongst the most popular and most researched MTD techniques [1], [2]. In IP shuffling MTD, also known as host address mutation MTD, network hosts are assigned a temporary IP address that is valid for only a certain period of time. These techniques are designed to counter network reconnaissance. While an attacker scans a network address range and proceeds with their attack based on this information, the MTD is triggered, and the IP addresses and any information about the hosts with a particular IP that were gathered by the attacker are no longer valid. Thus, the attacker must spend more resources and has a valid view of the network for only a limited time. In this work, we use an IP shuffling MTD based on Flexible Random Virtual IP Multiplexing (FRVM), where each port on each host is assigned a virtual IP address (vIP) [10]. Evaluations of this technique have shown it to be effective at reducing the attacker’s success in network reconnaissance through modeling [10], emulations [11], and in a realistic testbed [12].

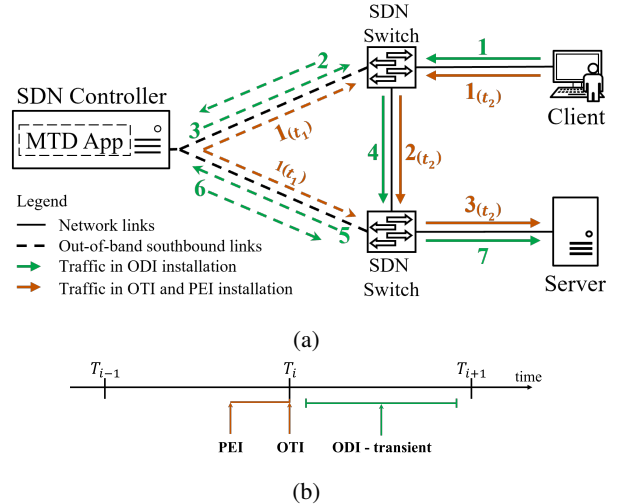


Fig. 1: (a) Steps of MTD update at T based on the update mechanism. Lines above the network connections (green) show the steps for on-demand installation (ODI), and the lines below the network connections (orange) show the steps for on-time installation (OTI) and PEI MTD, which happen at two distinct times t_1 and t_2 . (b) Timing of when the MTD update is made in the SDN switches for update T_i . In the ODI scheme, updates are made when a client connection is requested, making it transient. In PEI, updates are installed in advance but do not come into effect until T_i .

Each network host has a real IP address (rIP). The SDN controller assigns each host with virtual IP addresses and maintains a mapping between the rIPs and vIPs. When the MTD is triggered, new vIP addresses are assigned to the hosts. The controller adjusts the flow rules in the SDN-enabled switches to forward packets according to the mapped vIPs. The replacement of the rIP and vIP in the IP packets is done at the network edge, so the IP mutation is transparent to both end-user devices and network hosts. Legitimate clients can find the new vIPs through an authenticated DNS service. Any hosts scanning the network will not be aware that their previously discovered hosts are invalid when the IP shuffling has occurred.

The MTD triggering can be time-based, happening at regular intervals (which we call the MTD interval T), event-based, happening after some network event, such as an IDS alert, or a hybrid of the two. A shorter MTD interval will provide greater security against scanning attacks but also incur a greater performance cost [12].

III. MTD ARCHITECTURE AND IMPLEMENTATION

The MTD design defines the broad way the MTD should operate: when it should trigger, what should change in the system, and how it should change. The implementation details of how the MTD is triggered influence the fingerprint the operation leaves in the network traffic, which can then be exploited by an attacker. We call the sequence of events that are enacted at MTD trigger time the ‘update mechanism.’ Below,

we categorize and explore three possible update mechanisms and their characteristics.

A. On-Demand Installation

In on-demand installation (ODI), the controller removes any MTD-related rules in the switches at MTD trigger time. The switches always retain a rule to forward any packets that do not match any other rules to the controller. As such, when a new packet arrives at a switch with no corresponding rule, it is forwarded to the controller, who decides what to do. The MTD application in the controller computes what IP substitutions need to be made in the packet and installs a corresponding rule in the switches. Thus, MTD rules are installed on the switches when there is relevant traffic. The algorithm for this update mechanism is shown in Algorithm 1. The previous rules are deleted after the MTD trigger interval T has passed, and the rules are installed in each switch after a packet arrives at some other time. This typical mode of operation is described in [11].

Algorithm 1 MTD trigger algorithm for on-demand installation

```

SrcIP: the source IP address for traffic
DstIP: the destination IP address for traffic
Require:  $\Delta t \geq T$ 
switches  $\leftarrow$  deleteRules
updateDNS
Require: packet at switch
if Src or Dst is in Network then
  if Src is in Network then
    switch  $\leftarrow$  newRule(forward rIP to vIP
                          for DstIP)
  else if Dst is in Network then
    switch  $\leftarrow$  newRule(forward vIP to rIP
                          for SrcIP)
  end if
  Await confirmation from switch
  Forward packet along path to next switch
else
  Drop packet
   $\triangleright$  We can check for malicious or irrelevant traffic.
end if

```

This method mimics the common mechanisms that are used for maintaining routing tables and forwarding traffic on the internet, similar to updating addresses in layer-2 switches. The majority of the logic is independent for each switch and path, and the MTD operation interfaces only minimally with the normal flow of traffic. As a result, this method is very flexible and has minimal overhead. The flexibility means that, like in traditional networks, switches and hosts can be added at any time, and there is minimal impact on load balancing.

Some aspects of the operation of the MTD are also simpler in this scheme. Legitimate IP addresses do not need to be tracked in advance and instead can be checked at arrival time. Inserting rules has a performance cost, and once rules are added, the performance of a switch is inversely related to the

size of its flow table. In this scheme, rules are only installed when needed, so the impact on performance is minimized.

From a security perspective, on-demand installation allows the controller to try and distinguish between a legitimate client and an attacker at packet arrival time before installing the rules that allow access to network hosts. There are no additional components or databases needed to implement this method, and the details of the implementation are clear. Since the changes to the switches are dependent on the incoming traffic, the operations are naturally staggered in time, giving better resource utilization. However, this also means that there is no one singular MTD trigger time; rather, it is a transient period of time after T when the rules are most likely being installed. Based on our analysis of ODI with MTDsense, we propose on-time and pre-emptive installation, which we describe next.

B. On-Time Installation

In on-time installation (OTI), the switches are emptied and repopulated at the MTD trigger time T_i . As all of the MTD operations happen at the start of an interval, this is the only time that any fingerprints can be left in the network traffic. Once the installation period has passed, there will be no further MTD operations until the next trigger interval starts. The algorithm for this is shown in Algorithm 2. Once again, the rule to forward nonmatching packets to the controller is retained. In this scheme, any traffic that matches no rules will be forwarded to the controller, where it can be managed according to the firewall rules. In this case, the traffic will be dropped if the client cannot authenticate, or it will be forwarded through the network by the controller, similar to the ODI scheme.

Algorithm 2 MTD trigger algorithm for on-time installation

```

Require:  $\Delta t \geq T$ 
switches  $\leftarrow$  deleteRules
for all switch  $\in$  switches do
  for all client  $\in$  ExpectedClients do
    for all rIP  $\in$  rIPs do
      switch  $\leftarrow$  newRule(forward rIP to vIP
                            for DstIp)
      switch  $\leftarrow$  newRule(forward vIP to rIP
                            for SrcIp)
    end for
  end for
end for
updateDNS

```

This method ensures all new rules are installed at the switches some short time after the MTD trigger time. However, it is less flexible. Since the clients are not being discovered and authenticated at connection time, the controller must maintain a list of expected clients (both previously connected clients and those that could be expected to connect in this MTD interval). The controller needs to decide in advance which clients to allow and precalculate the paths through the network. The operations at MTD trigger time will also take longer as many rules need to be installed at once. In the case of event-based

MTD, using this mechanism also means there will be a longer delay when triggering the MTD after an event.

The lack of flexibility from this method means some downsides need to be mitigated. Since all clients are added at the same time this hinders the controllers ability to apply up to date load balancing, instead any load differences must be predicted in advance. Additionally, there may be a longer delay in making changes if the optimal path changes due to network events. These issues can be addressed at the next MTD trigger instead of at any time, and the MTD intervals may be short enough for this to be acceptable. Finally, in a large network, precalculating the paths can have a large storage overhead and require a lot of computing time.

C. Pre-Emptive Installation

The goal of the pre-emptive installation (PEI) method is to have the new rules come into effect at trigger time T_i , with any operations finalized by this time. Like in the OTI scheme, this again requires the controller to predetermine and authenticate clients. In this scheme, sometime before T_i , the controller installs the rules for the next MTD interval into the switches. The rules are configured to come into effect at the next MTD interval start time and expire at the interval end time. The algorithm for this scheme is shown in Algorithm 3.

Algorithm 3 MTD trigger algorithm for pre-emptive installation

```

Require:  $\Delta t < (T - \epsilon)$ 
for all  $switch \in switches$  do
  for all  $client \in ExpectedClients$  do
    for all  $rIP \in rIPs$  do
       $switch \leftarrow newRule(\text{forward } rIP \text{ to } vIP$ 
         $\text{for } DstIp, \text{ expire after } T)$ 
       $switch \leftarrow newRule(\text{forward } vIP \text{ to } rIP$ 
         $\text{for } SrcIp, \text{ expire after } T)$ 
       $\triangleright$  Rules are added at the bottom of the
        flow table.
    end for
  end for
end for
Require:  $\Delta t \geq T$ 
  updateDNS

```

In this way, the overhead of communication between the controller and the switches is taken on before the trigger time, so the actual delay for the change between the two sets of rules should be reduced. This requires that the switch allows the controller to install new rules without needing to reload or otherwise interfere with ongoing traffic. This method suffers from the same reduced flexibility as on-time installation.

In addition, there are security implications for PEI and, to some extent, OTI implementations. By preempting the MTD trigger and installing all the possibly required rules, the information about the configurations for the next MTD interval is available more completely and for a longer time. The full list of authenticated clients is also stored in two places and is now also being kept on the switches at all times.

Keeping in mind that SDN switches can also be compromised [13], this increases risk. Further, connecting devices cannot be re-assessed for access at connection time. These issues are bounded by the limited MTD interval, however additional access control mechanisms need to be implemented to manage dynamic changes during the interval.

Figure 1 shows a simplified model of the SDN environment, with the timings and behavior for the three update mechanisms. In the ODI case, all communication from the controller is made whenever traffic arrives at a switch. In the OTI and PEI cases, there is an initial time t_1 when all rules are installed and a later time t_2 when traffic arrives at the network. The difference in timing of the update with respect to the MTD interval is shown in Figure 1b.

D. Implementation and Data

1) *Environment and Component Behavior:* Data was collected in a small-scale SDN testbed, which offers the necessary dynamism and flexibility to effectively evaluate the performance and behavior of the MTD and the attacker. Because testbed experiments must run in real time and are therefore limited, we supplement the data with simulations using Mininet [14]. Mininet was carefully configured to simulate the testbed environment. These cover intermediate parameter values and allow for faster prototyping during development. The general structure of the network is shown in Figure 2. It includes the following components:

- **The controller**, which is connected to the switches through separate out-of-band links and implements the MTD. We use a Open Network Operating System (ONOS) 2.4.0 controller [15] and OpenFlow 1.3.
- **A DNS server**, which provides service to legitimate clients.
- **A router**, which divides the SDN network from the user network and acts to simulate a connection over the internet.
- **SDN-enabled switches** that are programmed by the controller. In the SDN-testbed, these are HP Aruba 2920F and 2930M network switches.
- **Servers** running an Apache webserver [16].
- **Clients** who periodically connect to the servers and request webpages of various sizes.
- **The attacker** which sits outside the SDN network and is connected to the same switch as the client.

2) *Threat Model:* The assumption of compromise is that the attacker needs to be in the same subnet as a legitimate client and be able to sniff the client's traffic. The attacker could be located anywhere in the network and could equally compromise the client, a switch between the client and the network, or any other elements on the path, but it does not need to (though it could) compromise a component inside the SDN network. The variety of choices means that it is more likely the attacker could achieve access to at least one of these components, and vulnerabilities are commonly found in various components of networks, including in the SDN components themselves [17]. The attacker passively sniffs the legitimate client's traffic, and since it is only sniffing passively and is

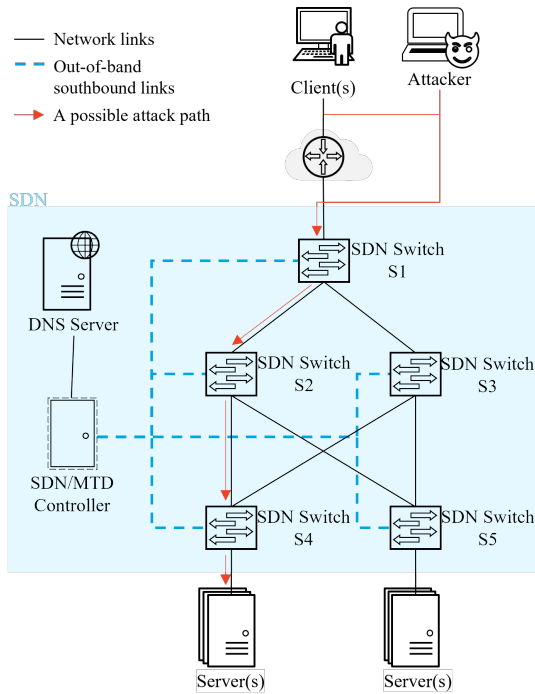


Fig. 2: Our experimental environment.

outside the protected network, it cannot be detected without extensive active scanning. We further relax the requirements by showing that the traffic need not necessarily come from one client but can be aggregated from multiple clients of the network. The data collection can be located anywhere, does not need to come from a singular source, does not involve active disruption to the network and so does not need to evade detection by the SDN network.

3) *Parameters and Data Collection*: Table I shows the different experimental parameters that were considered when taking data and the ranges of values that are considered for each of the variables. What follows is an explanation of each parameter and the reasoning for the chosen values.

The attacker observation window O_{win} is the duration of time the attacker sniffs network packets before trying to determine the MTD properties. This is varied between $3 \times \text{the mtd interval}$, which is guaranteed to capture 2 MTD triggers, and so is considered the lowest possible duration for calculating the interval, and 10 hours. These values are used to evaluate how long an attacker needs to observe a network to be able to deduce MTD properties with our method. The MTD interval T is the duration between two MTD triggers. The range here was chosen based on the analyses by Connell *et al.* [18], Mendonça *et al.* [19] and Kim *et al.* [12], who have analyzed the performance and security trade-off of the MTD interval which varies with respect to the expected attack length. Based on their analyses, we find an interval of 60-300s to be appropriate for our case and extend our window slightly in both directions to cover other cases.

Requests by the client are a Poisson process defined by the mean λ . The highest value for λ was chosen based on benchmarking of the testbed. Due to the limitation of the simulated network, higher values of lambda lead to occasional

TABLE I: Experimental Parameters and Their Values.

Variable	Notation	Range of values
Attacker observation window	O_{win}	$3T - 10$ hours
MTD interval (seconds)	T	30, 60, 120, 180, 240, 300, 600
# client requests/sec	λ	10, 5, 1, 1/2, 1/5, 1/15, 1/30, 2/T
# servers	n_S	1, 2, 3
# clients	n_C	1, 2, 3
Size of webpages	W	400kB - 4MB
Trials	N	30

network dropouts. The slowest client request interarrival time is half of the MTD interval, which is very slow considering the number of requests fielded by a typical server on the internet, but still ensures there is at least one request in each MTD interval, which is required for triggering in the ODI case. The lower bound of 10 requests per second is typical in the literature [12], [18]–[20] and less than the number of requests fielded by typical servers. For example, the Wikimedia Foundation reports an average of 400 requests per second per machine across their servers [21], and Google Cloud allows a maximum of 1000 concurrent requests per instance [22]. This makes it a reasonable minimum rate of traffic for the attacker to require.

Different numbers of servers and clients are used to demonstrate how the attack is affected by increasing the number of nodes in the network. The size of webpages available on the server ranges from 400kB to 4MB, following the typical page weights below the 75th percentile found on the internet as reported by the HTTP Archive [23]. The client requests webpages at random following a normal distribution with a mean file size of 2MB. Each set of experiments is run for 30 trials. For each trial, packets are sniffed from the network for the length of the observation window.

The exact set of parameters used for each experiment is specified in the results section. The data collected were the network packets sniffed by the attacker and the ground truth times when the MTD was triggered.

IV. ATTACKER'S APPROACH

The complete MTDSense pipeline is shown in Figure 3, and the details of specific steps are discussed below.

A. Feature Set

The features that are relevant to changes made by the MTD should capture changes in connections to a moving host in a network. These will appear across time and within flows rather than being distinguishable between individual packets. Therefore, both flow-based and window-based features were extracted from the raw packet data. The flow-based features are built on the CICFlowMeter-v4 feature extractor, which analyses flow traffic into 75 flow-level features [24]. Not all of these features are relevant to this problem, so those that were unchanged across all flows were removed at the outset. In addition to these, timing and delay features were added. These included the timing and delay of the TCP handshake and message body packets separately. The interarrival time between flows and entropy of IP addresses over time were also included, as these are expected to best directly capture

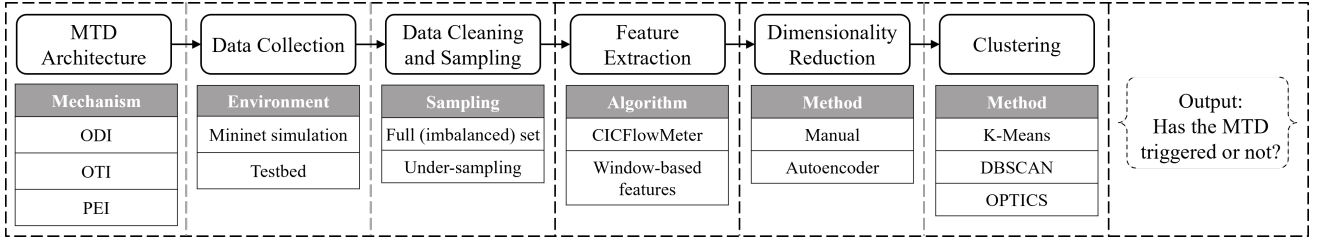


Fig. 3: Proposed MTDSense approach for data analysis.

the change to network components after an MTD trigger. An analysis of the importance of the relevance of the features is discussed in Section V-A4.

Naturally, most of the time, the MTD is not being triggered. This means the volume of traffic captured directly after an MTD trigger is far less than the volume of traffic in between triggers. Therefore, there is a large class imbalance when distinguishing between the two cases, which was addressed with undersampling.

B. Clustering Techniques

We assume that the attacker knows that the network is adopting an IP shuffling MTD but does not know the settings of the MTD and is not able to access the ground truth for the MTD trigger time or interval. As such, we must use unsupervised clustering algorithms to determine the MTD trigger time. To this end, we use primarily k -means clustering, which is a very commonly used method and continues to be used in some state-of-the-art clustering pipelines [25]. Other clustering algorithms, such as OPTICS, a generalization of the DBSCAN algorithm, which is density-based and can capture patterns of behaviors that k -means cannot, were also employed. However, after dimensionality reduction, differences in results were not pronounced.

Given that there is over 80 features and not all are relevant, we need a dimensionality reduction technique to obtain consistent results. The clustering techniques were combined with an autoencoder for dimensionality reduction. Firstly, this was done in two separate steps, where the autoencoder was trained and optimized for reconstruction loss, and the results were used to train the clustering algorithm. As an extension, the autoencoder and k -means algorithms were trained together with a combined loss function. This approach was introduced in [26] and is commonly used as a benchmark in clustering literature.

The state-of-the-art in unsupervised clustering is in the image domain and uses generative adversarial networks [27]. However, the image data is significantly different from our data and has more complex spatial relationships. These models do not transfer well to our domain. Additionally, we found little difference before and after fine-tuning, which indicates we have already extracted the maximum information from the data. Postprocessing methods are discussed where they are relevant.

C. Evaluation Metrics

Since the data is being clustered unsupervised, alternative metrics that do not rely on labeling need to be used. Comparing labels will lead to scores that are not well defined on different permutations of the clustering and give results that are difficult to compare. Instead, the metrics need to measure how well the data is grouped compared to the ground truth. For this, we use the standard metrics often used in benchmarking clustering algorithms. This includes the adjusted Rand index (ARI) [28] and the clustering accuracy [26] (which we report as ‘accuracy’). Note that ARI ranges mostly between 0 and 1; however, it does not change linearly in comparison to accuracy. The ARI will be 0 if the clustering is completely random and 1 if the cluster and class set are identical in their pairwise membership. The value is bounded by -0.5 , where negative values indicate the clustering is less in line with the truth labels than the expected random case.

V. EVALUATION

In this section, we evaluate the attacker’s ability to determine the timing of the MTD trigger with the following key questions. Firstly, how effectively can the attacker, with our proposed method, detect the MTD interval? Secondly, can modifying the method of triggering reduce the effectiveness of the detection? Thirdly, what conditions are needed for the attacker’s approach to be effective?

A. Determining That The MTD Has Been Triggered

The results in this section use data with the On-Demand Installation update mechanism, which we consider to be the most natural for a network to implement because it most closely conforms to the asynchronous behavior of the internet. Section V-B provides a comparison of attack results with other update mechanisms. The network configuration for this set of experiments comprised one server and one client, with data collected by the attacker and processed as described in Section IV to determine the network flows that follow an MTD trigger. In the case of the ODI update mechanism, this does not necessarily correspond to the absolute time the MTD triggered. Here, we report the MTD detection as successful for each item used in the ARI metric if the absolute time determined for the MTD trigger is within one second of the ground truth MTD trigger time, which is a small interval compared to the MTD interval T . We call this one second the ‘allowed detection delay’ s .

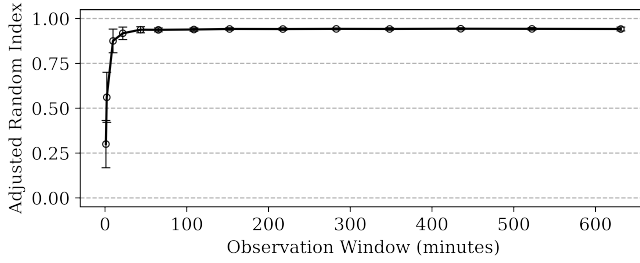


Fig. 4: The effect of attacker observation window on ARI for $T = 180s$.

The results demonstrate that the attacker can determine whether the MTD has been triggered with an ARI as high as 0.92 and accuracy as high as 0.97, depending on the experimental parameters.

The effect of network parameters on the fidelity of the results is explored below.

To determine the most appropriate observation window for the attacker to use, we tried possible values between one minute and 10 hours in an environment with an 180s MTD interval. In each case, the attacker collected data for the length of an observation window and clustered the data for that window. The results are shown in Figure 4, with each point being an average of 50 trials. Error bars show the 95% confidence interval. Based on this information, an observation window of 2 hours was used for all other results reported going forward, but note that the attacker’s information is fairly good with an ARI of 0.875 and accuracy of 0.94 for an observation window of 10 minutes, which is three times the MTD interval.

This shows that an attacker with a window as small as $3T$ could still gain useful information with a few trials, and the effectiveness plateaus at a window of two hours.

1) *The Effect of the MTD Interval:* Figure 5 shows the ARI in detecting whether the MTD triggered or not for different MTD intervals. From the results we can see that the detection rate is independent of the MTD interval, the attacker can determine T within this range with similar accuracy for all the values tested. Unlike the security benefit of MTD against traditional attacks that do not target the MTD mechanism directly, the accuracy of detecting that the MTD has triggered is not lowered by employing a lower MTD interval. As discussed in section III-D3, we exclude very short MTD intervals which we deem to have too large a performance cost. Although very short MTD intervals may lower the detection accuracy, they would not be performant and so are unlikely to be employed in a network.

We achieved a similar accuracy of detection when the MTD was triggered at a random interval.

For the final column of Figure 5 labeled ‘random’, the MTD interval was sampled randomly from a normal distribution with a range of 15 to 300 seconds at each trigger. The symptoms of the MTD triggering in the network are completely independent

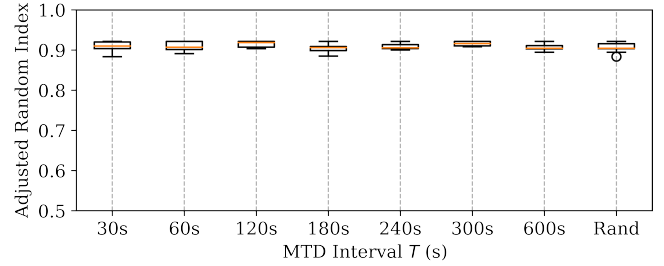


Fig. 5: The effect of varying the MTD interval T on the attacker’s achieved ARI. The right-most result is for an MTD where the next trigger time is decided by sampling randomly from a distribution.

of the cause of triggering and of T . Referring to the Algorithms 1-3, the effects are due to the ‘newRule’ installation steps and appear regardless of the condition that caused the MTD to trigger. For this reason, the method is still applicable to an MTD that is triggered by a random interval, and we predict it will also apply to an event-based MTD or any other triggering mechanism.

2) *The Effect of Client Traffic:* Figure 6 shows the effect of the average delay between client requests on the attacker’s ability to determine the MTD has triggered. The rate of client traffic ranges from 10 connections per second to one or two connections per MTD interval, as discussed in Section III-D3. Figure 6a shows the result over this full range, and 6b shows the results over a log axis so that the effect over the majority of the points is clearer.

Focusing on the $s = 1$ case, we can see the ARI is higher for lower delays until the lowest delay value, which resulted in less accurate detection than the second lowest value. This is due to the maximum throughput of our network and the delay between triggering the MTD within the network and updating the DNS server. Since the DNS update is not instantaneous, it is possible for a client connecting at the exact time between the update to receive the old vIP just before it becomes invalid and not be able to connect to it by the time they make their request to the network. The client-server connection will still be established since there is already a built-in timeout mechanism for requesting the address again and reconnecting.

However, this creates an additional performance degradation as the connection takes longer to establish. As the rate of requests increases, the chance of a request being made at the DNS update time increases, hence we see more of these events at the highest request rate. For the attacker determining that the MTD has triggered, this creates a different fingerprint in the traffic, and so the time is not detected accurately.

This means the ARI increases with increased rate of client requests until there are too many requests for the network to handle without some performance degradation, at which point the attacker’s accuracy also decreases.

The figure also shows the result for different allowed detection delays. The data must have ground truth values assigned to it. In the $s = 1$ case, a positive label is assigned

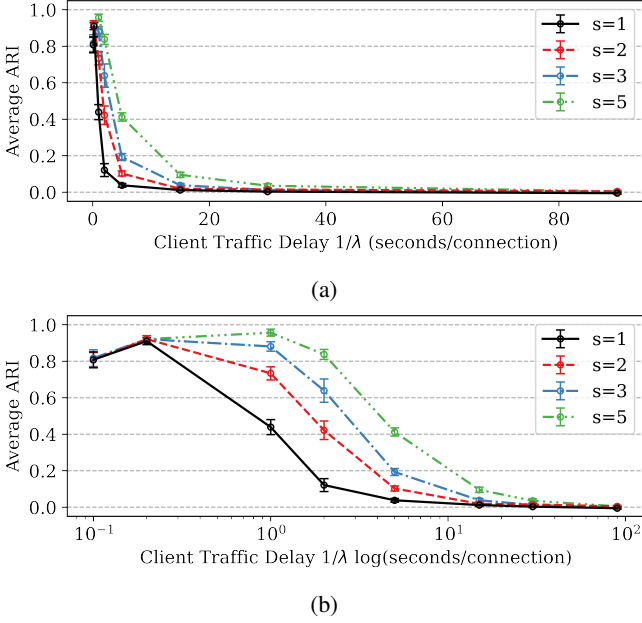


Fig. 6: (a) The effect of the rate of legitimate client traffic λ on ARI. In (b), the same data is shown over a log scale for clarity. Given over a range of allowed detection delays s .

to a flow if it is within one second of the MTD trigger. We can see that in the $s = 1$ case, the rate of client traffic must be higher than 5 requests/s to give good detection, and it starts to degrade at $\lambda = 1s$. Results for the other s values follow a similar pattern where $s = 2$ degrades at $\lambda = 2s$, and so on. Given the way the ground truth labels are assigned, this effect is less due to the attacker’s ability to detect the MTD trigger and more to do with the ODI update mechanism.

When using the ODI update mechanism, the MTD rules are not installed in the OpenFlow switches until there is relevant network traffic. Due to this, we can consider the MTD trigger time to be transient, where there is an initial phase at the trigger time T_i and a time when all the rules come into effect t_i^c , which is probabilistic. The time the rules are installed depends on when traffic arrives at a particular switch, so t_i^c is dependent on the rate at which traffic arrives, the number of paths through the network, and how those paths are utilized. The final rule can be installed at any time t_i^c before the start of the next interval T_{i+1} . Because the ground truth times are the trigger time T_i and the symptoms on the network depend on when the rules are installed t_i^c , the delay with which an attacker can determine that the MTD has triggered depends on the delay between when the MTD is triggered and when a client sends a packet. That is the attacker’s delay $\Delta t_A = \|\tilde{T}_i^A - T_i\|$ depends on the transient installation interval $\Delta t_I = \|t_i^c - T_i\|$. Since there is a causal relationship, the attacker’s delay cannot be shorter than the transient period ($\Delta t_A \geq \Delta t_I$). Note from the figure that as the rate of client traffic increases, the duration of the transient period decreases, so the attacker can accurately determine the MTD trigger time with a lower detection delay.

This delay due to the transient period is important because

TABLE II: The MTD Interval T As Predicted by the Attacker Based on Their Clustering.

Actual Interval (seconds)	Mininet Data		Testbed Data	
	Predicted Interval (seconds)	Error Rate	Predicted Interval (seconds)	Error Rate
60	59.73	0.45%	60.39	0.65%
180	179.58	0.23%	180.84	0.47%
300	300.32	0.11%	301.58	0.52%

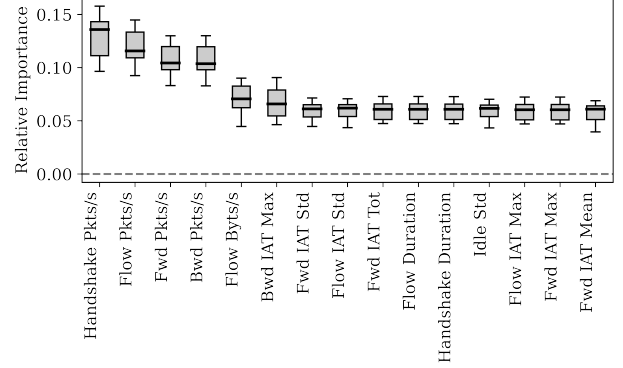


Fig. 7: Ranking of feature importance to clustering.

if the attacker wants to time their attacks to complete within an interval, they need to be able to accurately predict the duration when the current set of IP addresses is valid, and this duration is between the ground truth times T_i and T_{i+1} , with no fixed relationship to t_i^c . In the ODI case, the attacker can always tell whether the MTD has been triggered since some previous time, even when there is no traffic at the time of the MTD trigger. This includes the cases where there is only one connection or less in each MTD interval. The attacker can determine their confidence of when the MTD is triggered based on the rate of client traffic. This also means that in the ODI scheme, the attacker can calculate the MTD interval T accurately based on the timings of t_{ci} , even when there is not enough traffic to accurately calculate the time t_{Ti} . This is addressed in Section V-A3. The effect of the rate of traffic in the OTI and PEI MTDs is explored in the next section.

3) *Extracting The MTD Interval*: Given that the attacker knows when the MTD has triggered, they can restart their attack to maximize the possible attack window. However, if the attacker knows the MTD interval, they can further plan their attacks and limit them to those that will have the greatest chance of succeeding within the interval. To this end, we try to estimate the MTD interval T and report the accuracy with which it can be detected, given the accuracy of detecting the MTD trigger. The results are shown in Table II. Note that all values are predicted with an error of less than one percent.

4) *Feature Importance*: To rank the importance of the features to the clustering result, we use the approach proposed by Ismaili *et al.* [29], where a random forest classifier is trained with the clustered labels as the training labels, and the feature importance is extracted from the classifier. The results for the 15 most important features are shown in Figure 7.

We note that all of the most important features were related

to the timing features of the traffic, specifically to the rate of connections and the difference in rate between the TCP handshake and the rest of the connection. This indicates the MTD trigger is detectable due to the delay it creates in the network traffic when rules are installed. This suggests that detectability can be addressed by making the timing of the MTD more closely resemble normal network timings, which leads to the idea of alternative update schemes that aim to reduce the delay associated with the MTD trigger.

B. Effect of MTD Update Mechanism

We explored whether the other proposed update mechanisms affect the attacker’s ability to determine that the MTD has been triggered. Experiments were run using the OTI and PEI update mechanisms with an MTD interval of 180 seconds and for a range of client request frequencies λ . Each data point is the result of 30 two-hour trials, and the results can be compared to those in Figure 6. The results for the OTI mechanism are shown in Figure 8b. For convenience in comparison, the results for the ODI update mechanism over a smaller range are included as Figure 8a. We can see the attacker achieved a similar accuracy for both schemes when there was more client traffic (λ was high). As the amount of client traffic drops, the accuracy drops somewhat faster with OTI. In OTI, the rules are installed in the switch at the determined MTD time, which takes some slightly variable amount of time η . If the delay between the installation time and the arrival of client traffic is more than η , there will be no symptom of the MTD trigger in the network. However, at high λ where traffic is more likely to arrive within η , the attacker is still effective. Given that in a network, the utilization of each server is maximized as much as possible, this change may not be sufficient.

Also note that in the OTI case, there is no longer an effect due to s . At lower delays all s achieve similar accuracies, and at higher delays, all s have large overlapping error bars indicating the results are noise. This is because there the detection here is tied to T_i and not t_i^c , and therefore detection is not possible at lower rates of traffic.

Figure 9 shows the results for the PEI mechanism with the ODI and OTI cases overlaid for comparison. The attacker has almost no ability to determine the MTD has triggered here, even at higher λ . This indicates that the PEI mechanism, making updates before they are needed, is an effective way to reduce the timing difference before and after the MTD trigger and thus remove the attacker’s ability to detect the trigger. We note the detection here was variable, with some triggers being detected correctly while others were missed. In other words, there were more false negatives, but true positives were still present, while false positives were very low and true negatives high. The attacker is still able to detect some of the MTD triggers at high λ .

The theory for PEI aims to hide the MTD trigger entirely by performing all necessary changes preemptively, so there should be no symptoms of the trigger in the network. The results show some symptoms are still present, which we traced back to the timing of the network. While the MTD trigger is no longer responsive, there is still a delay in updating cached

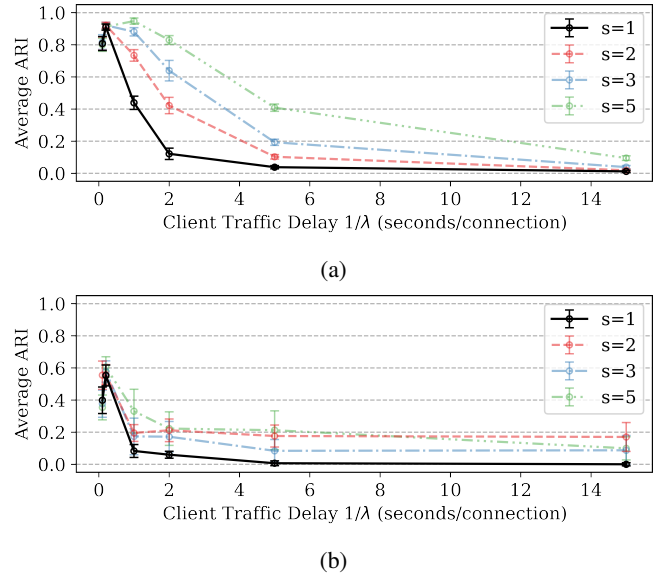


Fig. 8: ARI for clustering the MTD triggers with (a) the ODI update mechanism, and (b) the OTI update mechanism, $T = 180s$.

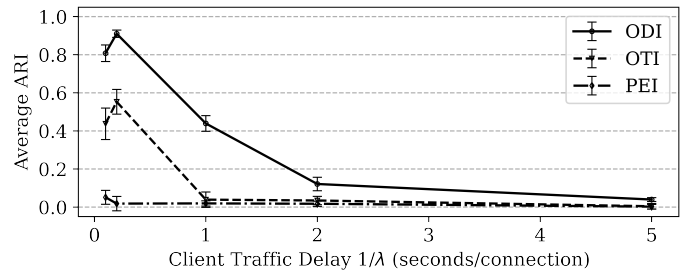


Fig. 9: Comparison of ARI for clustering the MTD triggers between all update mechanisms, $T = 180s$.

DNS records. Additionally, because updates are asynchronous, it is not always possible to synchronize all switches to update exactly at the same time. Given that the client requests can be so frequent, this means in some triggers, at least one switch still made a request to the controller.

The results show that we are able to minimize the range of amount of traffic the attacker’s approach is effective for with OTI and significantly reduce the detection with PEI.

In the PEI case, some symptoms are still present in the network, but the attacker’s detection is significantly dampened. When considering whether PEI can be deployed as a defense mechanism against this attack, we have to consider that there is a large performance cost associated with the non-asynchronous update mechanism (OTI and PEI). These performance overheads are detailed in Section 3. Given that there is already a performance cost associated with implementing OTI MTDs, which has to be contended with, this may not be desirable in a deployed network.

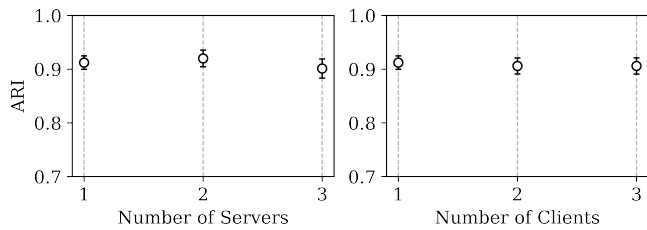


Fig. 10: The effect of the number of clients and servers on ARI when the total traffic volume across all hosts is kept constant.

C. Effect of Network Size

In this section, we first show that the total volume of client traffic, which makes the attack possible, need not necessarily come from one client or to one server. Under this model, we assume the attacker has compromised a series of devices that allow them to sniff the traffic from multiple clients to the network. The total rate of traffic from all clients λ_{tot} to all servers is kept the same but is generated by one, two, or three clients. The network utilizes an 180s ODI MTD. The results are shown in Figure 10.

The detection pipeline has to be expanded by one step in this scenario because, in the ODI scheme, flow rules are installed for each client and server pair at each switch. This means if we have n_c clients and n_s servers, after each MTD interval, $n_c \times n_s$ new rules are installed. Thus, the attacker will detect $n_c \times n_s$ MTD triggers that are all very close in time. When determining T_i , the attacker now groups all triggers it detects within two seconds as one MTD trigger, with the assumption that this is a far higher trigger frequency than would be expected of a network in practice.

The results show similar effectiveness across one client and server to multiple clients and servers. A minimum total rate of traffic from all clients to all servers is required for the attacker to be effective. However, allowing the attacker to collect this traffic across many clients relaxes the assumptions.

D. Cross Dataset Model Performance

The experimental results so far show that the attacker’s method for determining the MTD interval is effective across a range of MTD intervals in both a simulated and *testbed* environment. These results are achieved with 2 hours of training data. However, a key question to the practicality of this method is what the attacker can do if they do not have access to the network to collect enough training data. Additionally, what if the MTD is responsive, and so its settings change during or after the training interval? In this section, we show that trigger detection is effective across datasets.

Table III shows the performance of the attacker’s model with different training and testing environments using the ODI update mechanism. The data with the configuration in the row is used to train the autoencoder and clustered. Using the same autoencoder weights and cluster centers, the data from the configuration in the column is clustered, and the ARI is

TABLE III: Cross-dataset Performance of the Attacker Models. Results are given for both the simulation and testbed over different MTD intervals.

		Testing Dataset			
		Simulation T=60s	Testbed T=60s	Simulation T=180s	Testbed T=180s
Training Dataset	Simulation T=60s	0.917	0.92	0.922	0.914
	Testbed T=60s	0.918	0.916	0.911	0.892
	Simulation T=180s	0.921	0.897	0.912	0.918
	Testbed T=180s	0.899	0.906	0.913	0.923

calculated. As an example, the value in row 1 column 3 shows the result of training the model with data from a simulated network with 180s MTD and testing it against the testbed with 60s MTD. Across the diagonal is the performance of the models trained and tested from data in the same environment (though the training and test sets are separate). We can see from this table that the performance of the model does not drop significantly through this change. This is reasonable as the symptoms in the network traffic are not changed by changing the settings of the MTD, and as discussed in Section V-A4, the features that contribute most to the clustering are timing and rate-based features, which when normalized create the same pattern in the two environments.

This indicates that the attacker’s approach can still be successful if the settings of the MTD change and if the attacker cannot collect enough data from one set system.

So if one client cannot be completely monitored, the attacker can still be successful. Further, if the attacker still does not have enough access to the system but does know the overall design of the MTD, they can transfer a model from a different environment to the target environment. This may be applicable, for example, if the target is using a configurable commercial MTD. In such a case, the importance of ensuring the MTD is robust to this type of approach is very evident.

VI. RELATED WORK

Most literature on evaluating the effectiveness of MTDs has focused on assessing them against simple automated attackers that usually target traditional networks. Common types of attacks considered in network MTDs include scanning [30]–[32] and DDoS attacks [33]–[36]. While these works demonstrate the effectiveness of MTD techniques, the attacks considered are not tailored to an environment with MTD and assume the attacker has no knowledge that an MTD is being used.

Work considering an intelligent attacker that takes the MTD into consideration is very sparse. Jafarian *et al.* evaluated a combined MTD and deception technique using six skilled human attackers who were able to identify hosts using their fingerprints (open ports and running services), making the MTD alone much less effective [6]. However, they used a long MTD interval of 15 minutes and did not randomize port numbers. Additionally, this requires the full attention of skilled

human operators, which is costly for an attacker. Moghaddam *et al.* showed that if the attacker is assumed to know the MTD interval, they can plan their attacks to significantly increase the chances of success across all phases of a cyber attack [9]. However, they assumed the exact MTD interval is provided to the attacker, which is a strong assumption that is not realistic in typical attack environments.

There are also some works targeting MTD techniques directly in the domain of MAC address randomization MTDs in mobile phone devices. Vanhoef *et al.* uniquely identified and tracked mobile devices employing MAC address randomization MTD by clustering Wi-Fi probe requests [7]. This work demonstrated the ability to identify devices despite address mutation MTDs. However, it required that the protocol leak data by using non-standard headers and failing to change sequence numbers and scrambler seeds after mutation. In a similar study, Matte *et al.* identified mobile devices from Wi-Fi probes using only the timing of the probes, without the need to rely on leaked information [8]. These works circumvent the MTD instead of fingerprinting it but are notable in their use of clustering. To the best of our knowledge, no prior work has used machine learning or any other method to detect the MTD trigger and MTD interval in a comprehensive manner.

VII. LIMITATIONS

The main limitations of this work and possible areas for future research are as follows.

MTD dimension. MTD has three dimensions: 1) when to move, 2) what to move, and 3) how to move [2]. In this work, MTDSense was able to detect information about the first dimension: when to move. In our SDN setup, we have implemented other MTD techniques, such as virtual port shuffling and web application diversity, which will allow us to collect datasets involving the ‘what’ and ‘how’ aspects of MTD. In future work, we plan to extend the attacker’s scope to determine the other two dimensions of MTD using the same techniques.

Extension on client(s) traffic. The effectiveness of MTDSense is reliant on active clients in the network creating a minimum amount of traffic. This gives the attacker flexibility in what components are compromised aids in evading detection, however is ineffective if there is not enough client traffic. It is possible that some active scanning by the attacker can be incorporated in environments where client traffic is sparse. Moreover, our experiments are conducted with very small numbers of clients and servers. These should be scaled to larger environments, which have additional complexities.

Evaluation of other defense against MTDSense. We explore alternative mechanisms for implementing MTD as a possible defense. However, these do not completely hide the operation of the MTD and have associated performance costs. Other defenses are possible and need to be explored.

VIII. CONCLUSIONS

This work has presented a novel attacker approach named MTDSense. We have shown that by clustering network flows

eavesdropped from an SDN network, the attacker can determine when the MTD is triggered and calculate the MTD interval. To our knowledge, this is the first work to estimate the MTD trigger interval using an AI-based approach. The experimental results have shown this attacker approach is effective for a range of traffic volumes and network sizes, and the learning can be transferable between networks with differing configurations. Additionally, we have proposed two new MTD update mechanisms and evaluated them against MTDSense, showing that they are effective at low traffic volumes, although we predict that they have considerable performance and privacy costs. The attacker’s capability to determine the MTD interval significantly reduces the effectiveness of the defense and raises significant questions about the benefits of MTD.

REFERENCES

- [1] S. Sengupta, A. Chowdhary, A. Sabur, A. Alshamrani, D. Huang, and S. Kambhampati, “A survey of moving target defenses for network security,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1909–1941, 2020.
- [2] J.-H. Cho, D. P. Sharma, H. Alavizadeh, S. Yoon, N. Ben-Asher, T. J. Moore, D. S. Kim, H. Lim, and F. F. Nelson, “Toward Proactive, Adaptive Defense: A Survey on Moving Target Defense,” *IEEE Communications Surveys Tutorials*, vol. 22, no. 1, pp. 709–745, 2020.
- [3] SCIT Labs. (2024) SCIT: Our Technology. [Online]. Available: <https://www.scitlabs.com/products/>
- [4] NexiTech. (2023) Moving Target Defense. [Online]. Available: <https://nexitech.com/moving-target-defense/>
- [5] G.-I. Cai, B.-s. Wang, W. Hu, and T.-z. Wang, “Moving target defense: state of the art and characteristics,” *Frontiers of Information Technology & Electronic Engineering*, vol. 17, no. 11, pp. 1122–1153, 2016.
- [6] J. H. Jafarian, A. Niakanlahiji, E. Al-Shaer, and Q. Duan, “Multi-dimensional host identity anonymization for defeating skilled attackers,” in *Proceedings of the 2016 ACM Workshop on Moving Target Defense (MTD ’16)*, Vienna, Austria, 2016, pp. 47–58.
- [7] M. Vanhoef, C. Matte, M. Cunche, L. S. Cardoso, and F. Piessens, “Why MAC address randomization is not enough: An analysis of Wi-Fi network discovery mechanisms,” in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (ASIA CCS ’16)*, Xi’an, China, 2016, pp. 413–424.
- [8] C. Matte, M. Cunche, F. Rousseau, and M. Vanhoef, “Defeating MAC address randomization through timing attacks,” in *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec ’16)*, Darmstadt, Germany, 2016, pp. 15–20.
- [9] T. Moghaddam, M. Kim, J.-H. Cho, H. Lim, T. J. Moore, F. F. Nelson, and D. D. Kim, “A practical security evaluation of a moving target defence against multi-phase cyberattacks,” in *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. Melbourne, VIC, Australia: IEEE, 2022, pp. 103–110.
- [10] D. P. Sharma, D. S. Kim, S. Yoon, H. Lim, J.-H. Cho, and T. J. Moore, “FRVM: flexible random virtual IP multiplexing in software-defined networks,” in *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)*, New York, NY, USA, Aug. 2018, pp. 579–587.
- [11] C. Dishington, D. P. Sharma, D. S. Kim, J.-H. Cho, T. J. Moore, and F. F. Nelson, “Security and performance assessment of IP multiplexing moving target defence in software defined networks,” in *18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. Rotorua, New Zealand: IEEE, 2019, pp. 288–295.
- [12] M. Kim, J.-H. Cho, H. Lim, T. J. Moore, F. F. Nelson, and D. D. Kim, “Performance and security evaluation of a Moving Target Defense Based on a Software-Defined Networking Environment,” in *2022 IEEE 27th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2022, pp. 119–129.

- [13] S. Lee, C. Yoon, C. Lee, S. Shin, V. Yegneswaran, and P. A. Porras, "Delta: A security assessment framework for software-defined networks," in *The Network and Distributed System Security Symposium (NDSS)*, ser. NDSS '17, 2017.
- [14] Mininet Project. (2022, mar) Mininet - an instant virtual network on your laptop (or other PC). [Online]. Available: <https://mininet.org/>
- [15] Open Networking Foundation, "Open Network Operating System (ONOS) SDN Controller for SDN/NFV Solutions," 2024. [Online]. Available: <https://opennetworking.org/onos/>
- [16] The Apache Software Foundation, "The Apache HTTP Server Project," 2024. [Online]. Available: <https://httpd.apache.org/>
- [17] C. Yoon, S. Lee, H. Kang, T. Park, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Flow wars: Systemizing the attack surface and defenses in software-defined networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3514–3530, 2017.
- [18] W. Connell, D. A. Menascé, and M. Albanese, "Performance modeling of moving target defenses with reconfiguration limits," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 1, pp. 205–219, 2021.
- [19] J. Mendonça, J.-H. Cho, T. J. Moore, F. F. Nelson, H. Lim, A. Zimmermann, and D. S. Kim, "Performability analysis of services in a software-defined networking adopting time-based moving target defense mechanisms," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, Brno Czech Republic, 2020, pp. 1180–1189.
- [20] T. A. Nguyen, M. Kim, J. Lee, D. Min, J.-W. Lee, and D. Kim, "Performability evaluation of switch-over moving target defence mechanisms in a software defined networking using stochastic reward nets," *Journal of Network and Computer Applications*, vol. 199, p. 103267, 2022.
- [21] Wikimedia. (2024) Wikimedia Grafana Dashboards. [Online]. Available: <https://grafana.wikimedia.org/>
- [22] Google Cloud. (2024) Cloud Run Documentation: Maximum concurrent requests per instance (services). [Online]. Available: <https://cloud.google.com/run/docs/about-concurrency>
- [23] Web Almanac By HTTP Archive. (2024) Part IV Chapter 18 Page Weight. [Online]. Available: <https://almanac.httparchive.org/en/2020/page-weight>
- [24] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *4th International Conference on Information Systems Security and Privacy (ICISSP)*, vol. 1, Funchal, Madeira, Portugal, 2018, pp. 108–116.
- [25] A. M. Ikotun, A. E. Ezugwu, L. Abualigah, B. Abuhaija, and J. Heming, "K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data," *Information Sciences*, vol. 622, pp. 178–210, 2023.
- [26] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *International conference on machine learning*. PMLR, 2016, pp. 478–487.
- [27] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM Computing Surveys*, vol. 54, no. 2, pp. 38:1–38:38, 2021.
- [28] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical association*, vol. 66, no. 336, pp. 846–850, 1971.
- [29] O. A. Ismaili, V. Lemaire, and A. Cornuéjols, "A Supervised Methodology to Measure the Variables Contribution to a Clustering," in *Neural Information Processing*. Springer International Publishing, 2014, vol. 8834, pp. 159–166.
- [30] S. Achleitner, T. F. La Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy, and R. Chadha, "Deceiving network reconnaissance using SDN-based virtual topologies," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1098–1112, Dec. 2017.
- [31] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Spatio-temporal address mutation for proactive cyber agility against sophisticated attackers," in *Proceedings of the First ACM Workshop on Moving Target Defense (MTD '14)*, Scottsdale, Arizona, USA, 2014, pp. 69–78.
- [32] 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 *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 263–270.
- [33] Y. Zhou, G. Cheng, S. Jiang, Y. Hu, Y. Zhao, and Z. Chen, "A cost-effective shuffling method against DDoS attacks using moving target defense," in *Proceedings of the 6th ACM Workshop on Moving Target Defense (MTD '19)*, New York, NY, USA, Nov. 2019, pp. 57–66.
- [34] J. Steinberger, B. Kuhnert, C. Dietz, L. Ball, A. Sperotto, H. Baier, A. Pras, and G. Dreo, "DDoS defense using MTD and SDN," in *IEEE/IFIP Network Operations and Management Symposium (NOMS 2018)*. IEEE, 2018, pp. 1–9.
- [35] R. Meier, P. Tsankov, V. Lenders, L. Vanbever, and M. Vechev, "NetHide: secure and practical network topology obfuscation," in *27th USENIX Security Symposium (USENIX Security '18)*, Baltimore, MD, USA, 2018, pp. 693–709.
- [36] A. Aydeger, N. Saputro, K. Akkaya, and M. Rahman, "Mitigating cross-fire attacks using SDN-based moving target defense," in *Proceedings of the IEEE 41st Conference on Local Computer Networks (LCN)*. IEEE, 2016, pp. 627–630.