An Input-Aware Mimic Defense Theory and its Practice

Jiale Fu†, Yali Yuan‡∗, Jiajun He†, Sichu Liang§, Zhe Huang‡, Hongyu Zhu‡
†Southeast University, School of Mathematics. Emails: {213200900, 213203486}@seu.edu.cn
‡Southeast University, School of Cyber Science and Engineering. Email (*Corresponding): yaliyuan@seu.edu.cn
‡Southeast University, School of Cyber Science and Engineering. Emails: 213200037, 213200421@seu.edu.cn
§Southeast University, School of Artificial Intelligence. Emails: 213200709@seu.edu.cn

Abstract—The current security problems in cyberspace are characterized by strong and complex threats. Defenders face numerous problems such as lack of prior knowledge, various threats, and unknown vulnerabilities, which urgently need new fundamental theories to support. To address these issues, this article proposes a generic theoretical model for cyberspace defense and a new mimic defense framework, that is, Spatial-temporally heterogeneous, Input aware, and Dynamically updated Mimic Defense (SIDMD). We make the following contributions: (1) We first redefine vulnerabilities from the input space perspective to normalize the diverse cyberspace security problem. (2) We propose a novel unknown vulnerability discovery method and a dynamic scheduling strategy considering temporal and spatial dimensions without prior knowledge. Theoretical analysis and experimental results show that SIDMD has the best security performance in complex attack scenarios, and the probability of successful attacks is greatly reduced compared to the state-of-the-art.

1. Introduction

With the rapid development of networks, the popularity of the Internet has greatly facilitated our life. Meanwhile, network security issues receive increasing attention from the government and the academic world. However, existing network defense technologies, such as anti-virus, firewall, intrusion detection [1], honeypot [2], sandbox [3], and intrusion tolerance [4], can only handle attacks with known characteristics, which are essentially a “mending after a sheep is lost” methods.

In response to increasingly diverse cyberattacks, a large amount of research was conducted to propose effective defense technologies for specific types of attacks and application scenarios. For example, poisoning attacks, which affect the learning results by poisoning the training dataset, have been studied deeply for attack and defense against models, such as machine learning [11], federated learning [16]-[18] and so on. Usually, mining the characteristics of the model itself by separating the attack and defense modules to provide targeted defense is the main research approach at present. In other words, if there is more than one type of attack against the model, its defense performance is likely to decline significantly in the long term. Therefore, to address the limitation that a point-to-point defense strategy can only resist specific attacks, it is necessary to introduce mimic defense techniques to defend against multiple attacks in cyberspace.

Moving Target Defense (MTD) [5] is an active defense strategy and enables point-to-area defense. MTD enhances the uncertainty of the system by constantly changing the attack surface of the system so that attackers cannot accurately detect system vulnerabilities. To detect unknown vulnerabilities and backdoors, Jiangxing Wu proposed Cyber Mimic Defense (CMD) technology [6]. CMD is a dynamics, heterogeneity, and redundancy structure design based on the attributes of the system itself. Compared with MTD, CMD greatly increases the cost of attackers and gains higher robustness against multiple types of attacks. CMD theory has great potential in security performance, and many actual network scenarios are willing to pay more in exchange for reliable network security. In recent years, the mimic defense has been applied in Network Functions Virtualization (NFV) [7], cloud service [8], edge-computing [9], distributed system [10], etc., demonstrating a better defense effect.

Currently, research on mimic defense techniques mainly focuses on the impact of structure design, heterogeneous measurement, scheduling strategy, and adjudication mechanism on defense performance. The heterogeneity of mimic defense refers to the maximum heterogeneity of the hardware and software implementation of several executors capable of achieving the same function. The scheduling of mimic defense refers to the mimic scheduler dynamically updating several heterogeneous executors with a certain strategy and cleaning the expired executors offline. The scheduling strategy of mimic defense is an important foundation to ensure its dynamic and heterogeneous. A better scheduling strategy can make the mimic defense system expose fewer vulnerabilities and improve the security performance of the whole network.

However, in the real-world network security applications, the existing mimic defense has great shortcomings, mainly incorporating three points: (1) Although the simple definition of vulnerabilities brings great convenience for calculating the heterogeneity and similarity between executors, it is difficult to define the vulnerability for completely different executors in practice. (2) The heterogeneity of executors is static without considering the history attack knowledge. In practice, as the evaluation of attacks, new vulnerabilities
may appear, which are hardly taken into account in existing studies. (3) The current scheduling strategy of mimic defense only considers the common vulnerabilities among several executors in the same space, which ignores that among adjacent schedules. Without considering the temporal heterogeneity, it is possible to have a situation where “each schedule has few common vulnerabilities, but these common vulnerabilities remain unchanged over varied schedules”.

To solve the aforementioned issues, in this paper, we construct a theoretical model of mimic defense from an input perspective for the first time and propose a new mimic defense framework named SIDMD. We make the following contributions: (1) for a better heterogeneity measure, we give an exact definition of vulnerability that is understandable and easy to use in practice from the perspective of the input space, and introduce a k-order heterogeneity calculation method. (2) we design a dynamic update algorithm for executor vulnerabilities that can exploit historical attack information. (3) we first propose a dynamic scheduling strategy for executors based on spatiotemporal heterogeneity, which considers both temporal and spatial dimensions and significantly reduces the possibility that the system will expose vulnerabilities. (4) we run numerous simulations and compare SIDMD to state-of-the-art to verify its effectiveness. Besides, we test SIDMD on real-world applications, for example, resisting attacks of machine learning classification over a real-world dataset, to further prove the proposed model. We elaborate on our contributions below.

On the heterogeneous measurement, we note that the input space of heterogeneous executors is the same regardless of the differences in structure, implementation, etc. First, we abstract the input space of the executor into a metric space and define the concept of “distance” between inputs. Secondly, based on the above definition, we give the vulnerability set of executors and define the index for similarity and heterogeneity calculation, so that the normalized comparison between different executors is possible. Thirdly, we design a dynamic update algorithm for executor vulnerabilities based on Fitness Proportionate Sharing (FPS) clustering, which can update the summary of the input stream based on historical attack information, reflecting the adaptability to the dynamic environment and evolutionary attack.

On the scheduling strategy, we propose a novel scheduling strategy considering spatiotemporal heterogeneity. In addition to considering spatial heterogeneity, this strategy can combine temporal information and select the best executor to be replaced at the current time based on the distribution of executors’ vulnerabilities at the previous time. In this way, the potential harm can be minimized.

To evaluate SIDMD, first, we did numerous simulations. The results demonstrate that SIDMD performs well in discovering vulnerabilities, and the probability of being attacked by SIDMD is about 1/12 of the state-of-the-art. Besides, the total time of being attacked is about 1/14 of the state-of-the-art. The security performance of SIDMD with five online executors is similar to state-of-the-art with seven online executors. Second, we tested our model on a real-world application. The findings indicate that SIDMD is substantially more effective and robust than the state-of-the-art. Specifically, SIDMD consistently maintains a performance retention rate of 97% when defending against various attacks, which is greater than Proda [11] (96% at most) and TRIM’s rates [12] (90% at most), and SIDMD has good performance against different attacks, while the other two defense models have great differences in defense performance against different attacks. Additionally, the deployment of SIDMD does not reduce the models’ classification accuracy.

The rest of the paper is organized as follows. In Section 2, we provide the necessary background knowledge. In Section 3 we introduce our mimic defense theory. Section 4 provides the main framework of SIDMD. Subsequently, in Sections 5 and 6, we introduce the dynamic vulnerability update algorithm and the scheduling algorithm in detail, respectively. In Section 7, we analyze the complexity and cost of SIDMD and validate the defense performance of SIDMD through simulation and real datasets. Finally, we present related work in Section 8 and draw conclusions in Section 9.

2. Preliminaries

In this section, we introduce the main framework of traditional mimic defense and the necessary concepts and theorems.

2.1. The Main Framework of Mimic Defense.

The main framework of mimic defense is dynamic heterogeneous redundancy (DHR) [6]. As shown in Figure 1, it is mainly composed of five parts, including input agents, heterogeneous executor pool, scheduler, online executor set, and arbiter. The details for each of them are discussed as follows.

**Input agents**: The input agent replicates and distributes the input to each executor of the online executor set.

**Heterogeneous executor pool**: The heterogeneous executor pool includes a variety of functionally equivalent and structurally different heterogeneous executors. The heterogeneous executor pool is an alternative pool in which executors...
can be scheduled into the online execution set and process input.

**Scheduler:** The scheduler selects a certain number of executors from the heterogeneous executor pool to be online with a certain scheduling algorithm.

**Online executor set:** An online executor set is a collection of heterogeneous executors running online, where each executor is independent and processes input data in parallel. Their outputs are aggregated to the arbiter.

**Arbiter:** The arbiter will aggregate the outputs of the online executors and generate voting results using a specific voting algorithm.

The heterogeneity of the executors in the heterogeneous executor pool ensures the heterogeneity of the DHR. The scheduler continuously schedules over time to ensure the dynamism of the DHR, and the online executor set ensures the redundancy of the DHR. The dynamism, heterogeneity, and redundancy of DHR make the system uncertain in time and space, making it difficult for attackers to exploit the vulnerabilities of the system so that the system can obtain endogenous security features.

### 2.2. Metric Space

We deploy in our work the metric space, so we recall here the definition of metric space.

**Definition 2.1.** Let $A$ be a set, and let $d(\cdot, \cdot)$ be a binary real-valued function on $A$. We say $d(\cdot, \cdot)$ a metric on $A$, if $d(\cdot, \cdot)$ satisfies

1. $d(a, b) \geq 0, \forall a, b \in A$, and $d(a, b) = 0 \iff a = b$;
2. $d(a, b) = d(b, a), \forall a, b \in A$;
3. $d(a, b) \leq d(a, c) + d(c, b), \forall a, b, c \in A$.

Obviously, the metrics on a set are not unique, and we need to define different metrics according to different scenarios.

### 2.3. Equivalence Relation and Set Partitioning

We deploy in our work the equivalence relation and set partitioning related theory, so we recall here the relevant theory.

**Definition 2.2.** Let $A$ be a set, and let $R$ be a binary relation on $A$. We say $R$ an equivalence relation on $A$, if $R$ satisfies

1. $aRa$;
2. $aRb \Rightarrow bRa$;
3. $aRb, bRc \Rightarrow aRc$.

where $a, b, c \in A$.

**Theorem 2.1.** An equivalence relation on set $A$ uniquely determines a partition of set $A$, which is there is a unique set of sets $A_1, A_2, \ldots, A_n \subset A$, s.t.

1. $A = A_1 \cup A_2 \cup \cdots \cup A_n$;
2. $A_i \cap A_j = \emptyset, \forall i, j$;
3. $aRb \iff a, b \in A_i, \exists i$.

### 3. Input-Aware Mimic Defense Theory

In this section, we assume the input space of the executors to be a metric space. By defining the equivalence relation for the abnormal input set of the executor, we define the vulnerability of the executor, and then we considered the high-order similarity and heterogeneity of executors to better detect vulnerabilities in practice.

It is challenging to define the similarity and heterogeneity of the executors from the perspective of the executors themselves due to their differences. We note that their input space is always the same, regardless of how the two executors differ from one another. Therefore, from the viewpoint of the input space, executors can be contrasted in a normalized manner. On the basis of this idea, we offer the theoretical model below that describes the similarity and heterogeneity of executors.

**Definition 3.1.** Let $m$ denote an input to the executors and let $M = \{m\}$ denote the set of all inputs, we also call $M$ the input space of executors.

In real-world applications, there are many different executors. Typically, all of our attention is focused on the output that the executor will produce for a given input and the accuracy of that output. Therefore, we can abstract the executors as a function with the input as an argument.

**Definition 3.2.** Let function $f$ denote an executor, and let $F$ denote the set of all executors. Let $f(m)$ denote the output of executor $f$ for input $m$.

To state our results more conveniently, we introduce a perfect executor.

**Definition 3.3.** Let function $f_0$ denote a perfect executor, which means that for any given input $f$, $f_0$ will always output correctly.

It can be seen that the result of $f_0(m)$ represents the output that the executor $f$ should produce, in other words, the correct output. But in reality, since each executor always has known or unknown vulnerabilities, $f_0$ is only an ideal model and has only theoretical significance. And we cannot know the value of $f_0(m)$. The perfect executor $f_0$ is introduced here only for the convenience of presentation.

**Definition 3.4.** Let $m$ be an input and let $f$ be an executor. We say $m$ is a normal input of executor $f$, if $f(m) = f_0(m)$. We say $m$ an abnormal input of executor $f$, if $f(m) \neq f_0(m)$.

**Definition 3.5.** We say an input $m$ is a known input if $m$ is what people find in practical applications. Let $M_s$ denote a known input and let $M_s = \bigcup \{m_s\}$ denote the set of all known inputs.

Obviously, $M_s$ satisfies

$$M_s \subset M.$$  \hspace{1cm} (1)

And in practice, $M$ is usually an infinite set, and $M_s$ is usually a finite set.
Definition 3.6. Let \( f \) be an executor, and let \( V(f) \) denote the set of all abnormal inputs of \( f \) which is
\[
V(f) = \{ m \in M \mid f(m) \neq f_0(m) \}. \tag{2}
\]

Definition 3.7. Let \( f \) be an executor, and let \( V_s(f) \) denote the set of all known abnormal inputs of \( f \) which is
\[
V_s(f) = V(f) \cap M_s. \tag{3}
\]

Assumption 3.1. Let \( m_1, m_2 \) be two abnormal inputs of executor \( f \). If \( m_1 \) and \( m_2 \) are similar enough, that is, \( d(m_1, m_2) \) is less than a given small positive number \( \varepsilon_0 \), then we can think that the abnormal input \( m_1 \) and \( m_2 \) are exploiting the same vulnerability of \( f \).

In accordance with Assumption 3.1, we propose a definition of executors’ vulnerability based on set partitioning and equivalence relations.

Definition 3.8. Let \( A \) be a subset of the input space \( M \), let \( a, b \) be two elements in \( A \), let \( d(\cdot, \cdot) \) be a metric on \( M \), we say that \( a \) and \( b \) are related, denoted as \( a \sim b \), if there are some inputs \( m_1, m_2, \ldots, m_n \) satisfy
\[
1) \ d(a, m_1) < \varepsilon_0, d(m_n, b) < \varepsilon_0; \\
2) \ d(m_i, m_{i+1}) < \varepsilon_0, i = 1, 2, \ldots, n - 1; \\
3) \ m_1, m_2, \ldots, m_n \in A.
\]
where \( \varepsilon_0 \) is a small positive number.

Obviously, " \( \sim \) " has the following properties.
\[
1) \ a \sim a; \\
2) \ a \sim b \Rightarrow b \sim a; \\
3) \ a \sim b, b \sim c \Rightarrow a \sim c.
\]
where \( a, b, c \in A \). Then, " \( \sim \) " is an equivalence relation on set \( A \).

Definition 3.9. Let \( f \) be a executor, let \( V(f) \) be the set of all abnormal inputs of \( f \), and let " \( \sim \) " be the equivalence relation mentioned in Definition 3.8. According to Theorem 2.1, based on this equivalence relation, we can uniquely divide \( V(f) \) into some disjoint subsets, and we say each subset is a vulnerability of the executor \( f \).

After defining the vulnerabilities of the executors, we consider the similarity and heterogeneity of the executors.

Definition 3.10. Let \( A \) be a subset of \( M \). Let \( P(A) \) represent the probability of drawing a sample from \( M \) that happens to belong to \( A \).

Obviously, the higher the probability that \( f_1 \) and \( f_2 \) are wrong at the same time, the more similar \( f_1 \) and \( f_2 \) are. So the similarity of \( f_1 \) and \( f_2 \) can be evaluated as the probability that \( f_1 \) and \( f_2 \) output the wrong result at the same time, i.e. \( P(V(f_1) \cap V(f_2)) \), the heterogeneity of \( f_1 \) and \( f_2 \) can be evaluated as \( 1 - P(V(f_1) \cap V(f_2)) \). Next, we discuss the similarity and heterogeneity of executors in general.

Definition 3.11. Let \( f_1, f_2, \ldots, f_n \) be \( n \) executors, and the \( k \)-order similarity of these \( n \) executors can be evaluated as: for a random input, the probability that \( k \) of the \( n \) executors make mistakes at the same time, and denoted as \( P_k(f_1, f_2, \ldots, f_n) \). \( P_k(f_1, f_2, \ldots, f_n) \) can be calculated as
\[
\begin{align*}
P_k(f_1, f_2, \ldots, f_n) &= \sum_{i=1}^{C_n^k} P(A_i) - \sum_{1 \leq i < j \leq C_n^k} P(A_i \cap A_j) \\
&\quad + \sum_{1 \leq i < j < k \leq C_n^k} P(A_i \cap A_j \cap A_k) - \cdots \\
&\quad + (-1)^{C_n^k-1} P(A_1 \cap \cdots \cap A_{C_n^k}),
\end{align*}
\]
where each \( A_i \) represents an intersection of some \( k \) executors’ abnormal input sets.

\[
A_i = \bigcap_{j=1}^{k} V(f_j), \tag{5}
\]
where \( 1 \leq j \leq n \) and \( i_a \neq i_b, \forall a, b \).

The reason why we define the high-order similarity of multiple executors is that it can directly reflect the probability of arbitration failure. For example, if a voting-based arbitration DHR model deploys \( 2l+1 \) executors, then the probability of the entire DHR model failing is equal to the probability of \( l+1 \) executors failing at the same time. Therefore, our method on the definition of high-order similarity can well evaluate the defense performance of DHR models. Naturally, \( k \)-order heterogeneity of executors can be evaluated as \( 1 - P_k(f_1, f_2, \ldots, f_n) \).

Obviously, to evaluate the high-order similarity and heterogeneity of multiple executors, the key is to calculate the value of the function \( P(\cdot) \). However, in practical situations, it is difficult for us to obtain the abnormal input set \( V(f) \) of the executor \( f \) and the input space \( M \). That is to say, it is almost impossible for us to get the value of \( P(\cdot) \) by direct calculation. Therefore, we propose an approximate calculation of the function \( P(\cdot) \) based on the known input set \( M_s \).

Theorem 3.1. Assuming that the elements in \( M_s \) are all extracted independently and identically from \( M \), then for any subset \( A \) of \( M \), \( \forall \varepsilon > 0 \) satisfy
\[
\lim_{|M_s| \to \infty} Pr(\lfloor \frac{|M_s \cap A|}{|M_s|} \rfloor - P(A) < \varepsilon) = 1, \tag{6}
\]
where \( Pr(\cdot) \) denotes the probability of something happening, and \( \lfloor \cdot \rfloor \) denotes the number of elements in a set.

Proof. This theorem is easy to be proved by Bernoulli’s law of large numbers.

In accordance with Theorem 3.1, under the condition that the known inputs are independent and identically distributed, for any given subset \( A \) of \( M \), we can approximate the true probability \( P(A) \) with \( \lfloor \frac{|A \cap M_s|}{|M_s|} \rfloor \). We denote \( \lfloor \frac{|A \cap M_s|}{|M_s|} \rfloor \) with \( P'(A) \). Next, we introduce a method for finding an approximation \( P'_k(f_1, f_2, \ldots, f_n) \) of the \( k \)-order heterogeneity \( P_k(f_1, f_2, \ldots, f_n) \) based on the known input space \( M_s \).

Let \( f_1, f_2, \ldots, f_n \) be some executors. A straightforward way to calculate the k-order similarity of executors
\[ P_k'(f_1, f_2, \ldots, f_n) \] is just like Equation (4), and just replace \( V(f_i) \) with \( V_s(f_i) \). But such a definition is unreasonable. Even if two executors \( f_1 \) and \( f_2 \) are sufficiently similar, the intersection of \( V_s(f_1) \) and \( V_s(f_2) \) is probably going to be an empty set because the input space of the executors is frequently continuous and the known input space is only a finite number of points sampled from the input space. To address this, we provide a novel approach to computing intersections. We loosen the requirements for the intersection operation’s elements, meaning that as long as two elements are sufficiently similar to one another, they can still be considered intersection operation elements.

**Definition 3.12.** Let \( A_1, A_2, \ldots, A_n \) be some sets, and we say \( A \in \bigcup_{i=1}^{n} A_i \) is a common element within \( \varepsilon \)-error tolerance of \( A_1, A_2, \ldots, A_n \), if

\[ \forall i = 1, 2, \ldots, n, \exists a_i \in A_i \text{ s.t. } d(a_i, a) < \varepsilon. \tag{7} \]

Let \( (A_1', A_2', \ldots, A_n')_0 \) denote the set of all \( a \in \bigcup_{i=1}^{n} A_i \) that satisfies the above condition, and we say \( (A_1', A_2', \ldots, A_n')_0 \) is the intersection of \( A_1, A_2, \ldots, A_n \) within \( \varepsilon \) error tolerance.

The commutative law is easily verified to be satisfied by the set operation, hence the outcome of the operation has nothing to do with the set’s order. The associative law, however, is not satisfied by this operation, thus it cannot be calculated by breaking it up into smaller parts and then synthesizing them.

In the case of no misunderstanding, we write \( (A_1', A_2', \ldots, A_n')_0 \) as \( A_1', A_2', \ldots, A_n' \) for convenience. The known common abnormal input set can be defined as follows in accordance with Assumption 4.1.

**Definition 3.13.** Let \( f_1, f_2, \ldots, f_n \) be some executors, and \( V_s(f_1), V_s(f_2), \ldots, V_s(f_n) \) are their known abnormal input set. We say \( m \in \bigcup_{i=1}^{n} V_s(f_i) \) is a common abnormal input if \( m \in V_s(f_1), V_s(f_2), \ldots, V_s(f_n) \), and we say \( V_s(f_1), V_s(f_2), \ldots, V_s(f_n) \) is a common abnormal input set of executors \( f_1, f_2, \ldots, f_n \).

Based on this, we can evaluate k-order similarity and heterogeneity of executors by define \( P_k'(f_1, f_2, \ldots, f_n) \) as

\[
P_k'(f_1, f_2, \ldots, f_n) = \sum_{i=1}^{C_n^k} P'(A_i) - \sum_{1 \leq i < j \leq C_n^k} P'(A_i \cdot A_j) + \sum_{1 \leq i < j < k \leq C_n^k} P'(A_i \cdot A_j \cdot A_k) - \cdots + (-1)^{C_n^k-1} P(A_1 \cdot A_2 \cdot \cdots A_{C_n^k}),
\]

where each \( A_i \) represents an intersection of some \( k \) executors’ known abnormal input sets.

\[ A_i = V_s(f_{i_1}) \cdot \cdots \cdot V_s(f_{i_k}), \tag{9} \]

where \( 1 \leq i_j \leq n \) and \( i_a \neq i_b, \forall a, b. \)

Naturally, the k-order heterogeneity of executors can be calculated as \( 1 - P_k'(f_1, f_2, \ldots, f_n) \).

4. An Overview of SIDMD Framework

In this section, we introduce the main framework of the SIDMD model, and a general algorithm description of the SIDMD framework is presented in Algorithm 1. Based on theoretical models in Section 3, we propose the framework of SIDMD: spatiotemporally heterogeneous, Input-aware, and dynamically updated mimic defense, and the framework of SIDMD is shown in Figure 2.

First, the real-time input information will be judged as to whether it is an abnormal input for an executor. If so, it is saved and used to update the known abnormal input set of the executor. When an executor has saved enough abnormal inputs, these saved abnormal inputs are given as input chunks and summarized by the clustering algorithm. The summary of the input chunk \( CHS \) will then be used to update the summary of the input stream \( ISS \). Finally, When scheduling, the scheduling algorithm considers the factors of time and space and obtains the current optimal scheduling strategy according to the historical scheduling strategy and the summary of the input stream.

Let \( IS \) be an input stream and \( CH_i \) be a chunk of input from \( IS \) such that \( IS = \bigcup_{t=1}^{\infty} \{ CH_t \} \), where \( t \) refers to the time index of a input chunk. For each input chunk, \( CH_t = \bigcup_{t=1}^{\infty} \{ m_{t,i} | m_{t,i} \in M \} \), where \( m_{t,i} \) denotes a input sample in \( CH_t \). Let \( CH_t \) denote the abnormal input of the \( j \)th executor in \( CH_t \). Let \( IS \) represent the abnormal input stream of executor \( i \), where \( IS^i = \bigcup_{t=1}^{\infty} \{ CH_t \} \). Let \( CHS^i_t = \{ C^j_{CH_t}, Z^j_{CH_t} \} \) be the summary of \( CH_t \), where \( C^j_{CH_t} \) represents the center of the \( j \)th cluster in \( CH_t \), and the \( Z^j_{CH_t} \) represents the number of elements in the cluster. Let \( ISS_t^i \) be the summary of \( IS^i \) up to time \( t \), and the definitions of \( C^j_{ISS_t^i}, Z^j_{ISS_t^i} \) are similar to the above. Let \( SS_t \) represents the scheduling strategy at time \( t \). Assume that there are a total of \( N_p \) executors in the heterogeneous executor pool.

5. Dynamic Discovery of Vulnerabilities

In this section, we describe how to dynamically discover vulnerabilities in real-world. The dynamic discovery of vulnerabilities can be divided into two parts: obtaining abnormal inputs for each executor in real-world and automatically updating vulnerabilities over time. Given that each executor tends to have an infinite number of abnormal inputs over time, the dynamic update algorithm we propose only needs to save a summary of each cluster, greatly reducing the algorithm’s space complexity.

5.1. Obtaining Abnormal Inputs

It can be challenging to determine in advance whether an input is normal or aberrant in practice. As a result, we
Algorithm 1 An overview of SIDMD framework

Input: IS; r: the radius of the cluster; r₀: maximum distance to merge clusters.

Output: SS

1: for t = 1 to ∞ do
2: if ISSᵢᵗ ≠ ∅, ∀ᵢ then
3: Use the scheduling algorithm to obtain the scheduling policy SSᵢᵗ
4: else
5: Random schedule to get scheduling policy SSᵢᵗ
6: end if
7: for i = 1 to N_p do
8: Judge the abnormal input of the i-th executor in CHᵢᵗ and get CHᵢᵗ's summary CHSᵢᵗ
9: if ISSᵢᵗ = ∅ then
10: ISSᵢᵗ = CHᵢᵗ
11: else
12: Use CHSᵢᵗ to update the ISᵢᵗ's summary up to time t − 1 ISSᵢᵗ₋₁ to obtain ISSᵢᵗ
13: end if
14: end for
15: end for

suggest a technique that may automatically assess whether the input is normal to an executor using SIDMD’s arbitration mechanism.

For the input information, each executor will output a result, and these results will be combined to produce the arbitration mechanism’s output. We compare the output of each executor with the output of the arbitration. It is deemed to be correctly outputting if the executor’s result matches that of the arbiter; otherwise, it is deemed to be wrongly outputting. Algorithm 2 provides a full description of the process.

Remark 5.1. Although the outcome of the arbitration may not always be accurate, the probability that it is correct is substantially higher than the probability that a single executor will get the right answer. Therefore, it is feasible to judge the abnormal input of the executor in this way.

Algorithm 2 Obtaining abnormal inputs

Input: m: input; {f₁, f₂, …, fₙ}: online executor set; Arbiter

Output: 0-1 vector b, whose i-th component represents whether m is an abnormal input of the i-th executor

1: for i=1 to n do
2: Result(i) = fᵢ(m)
3: end for
4: Result₀ ← Arbiter(Result)
5: for i=1 to n do
6: if Result(i) = Result₀ then
7: b(i) ← 0
8: else
9: b(i) ← 1
10: end if
11: end for

5.2. Dynamic Update Algorithms

Each executor’s vulnerabilities change over time in real-world applications. New vulnerabilities will inevitably arise as a result of the attacker’s constant progress and the executor’s wear and tear. Inspired by Yan et al. [19], we propose an algorithm to update the executor’s vulnerability set in accordance with real-time aberrant input information. The detailed steps are listed below.
First, set parameters $r$ and $N$, for each $CH_t$, and randomly select $N$ candidate centers from $CH_t$ and evaluate their density as their fitness. With the Gaussian kernel type of density function used in [19], the fitness of each candidate center is defined as follows.

$$f(x_i^t) = \frac{\sum_{m=1}^{N} (e^{-\frac{d(x_i^t, x_m^t)}{\sigma}})^\gamma, i \in [1, N]}{\sum_{i=1}^{N} f(x_i^t)}.$$  \hspace{1cm} (10)

where $\sigma$ denotes the variance of data, and $\gamma$ denotes the scaling parameter that approximates the shape of clusters. $N$ is the number of candidate centers. And $d(x_i^t, x_m^t)$ represents the metric between $x_i^t$ and $x_m^t$.

**Remark 5.2.** The reason why $N$ candidate cluster centers are selected from $CH_t$, rather than the entire $CH_t$ as a candidate cluster center, is to prevent the computational complexity from being too high. Therefore, when the number of elements in the input chunk is relatively small, the entire a can also be used as a candidate cluster center.

Next, we choose the input with the highest density value as the center of the first cluster. Then, the input distances less than $r$ from the center of the cluster are taken as the elements in the first cluster, and fitness proportionate are selected from $CH_t$ as the center of the first cluster. Then, the input distances from the center of the cluster are taken as the elements in the first cluster, and fitness proportionate are shared among them. The new fitness (density) is evaluated as follows.

$$f(x_i^{tj})_{new} = \frac{f(x_i^{tj})_{old} \sum_{j=1}^{n_i} f(x_i^{tj})_{old}}{\sum_{j=1}^{n_i} f(x_i^{tj})_{old}},$$  \hspace{1.3cm} (11)

where $x_i^{tj}$ denotes the $j$th element in the $i$th cluster at time $t$. $f(.)_{new}$ and $f(.)_{old}$ denote the new fitness and old fitness.

Then, we choose the input with the highest fitness at this time as the center of the second cluster, and so on, until all candidate cluster centers belong to a certain cluster. The pseudo-code for this process is provided in Algorithm 3.

Next, we introduce how to dynamically update the summary of an executor. Assuming that the summary at time $t + 1$ is $CH_{St+1}$, and up to time $t$, the summary of the input stream is $ISS_t$. We set the parameter $r_0$, and assume that when the distance between the two cluster centers is less than $r_0$, they should be the same cluster, that is, the difference between the two cluster centers is considered to be caused by random factors. The way to update $ISS_t$ is as follows.

For each cluster in $CH_{St+1}$, denoted as $C_{CH_{St+1}}^{ij}$, if there are some clusters in $ISS_t$, denoted as $C_{ISS_t}^{ij}$, satisfying $d(C_{CH_{St+1}}^{ij}, C_{ISS_t}^{ij}) < r_0$, they will be merged into one cluster. Suppose the merged cluster is $C_{ISS_{t+1}}^{ij}$ and $Z_{ISS_{t+1}}^{ij}$ is its density, then the specific update formula is as follows.

$$\begin{cases} C_{ISS_{t+1}}^{ij} = C_{CH_{t+1}}^{ij} \\ Z_{ISS_{t+1}}^{ij} = Z_{CH_{t+1}}^{ij} + \sum_{j=1}^{|ISS_t|} C_{ISS_t}^{ij} \mathbb{I}(d < r_0) \end{cases} \hspace{1cm} (12)$$

where $\mathbb{I}(\cdot)$ denotes indicative function and $d$ denotes $d(C_{CH_{St+1}}^{ij}, C_{ISS_t}^{ij})$. It takes value 1 when the event happens and value 0 when the event does not happen. The remaining clusters either belong only to $CH_{St+1}$ or only to $ISS_t$, we add it to the new cluster $ISS_{t+1}$. The pseudo-code for this process is provided in Algorithm 4.

In order to ensure the vulnerability update capability of SIDMD, we can regularly normalize the density of all clusters, and delete clusters with small density. This not only adds a forgetting mechanism to SIDMD to ensure its update capability but also prevents a cluster from being too large to save. Based on this, the calculation of k-order similarity and heterogeneity between executors only needs to turn the original count into a weighted sum, where the weight is the density of the cluster center. The value $P'(A)$ is calculated as follows.

$$P'(A) = \frac{\sum_i Z_i \mathbb{I}(C_i \in A)}{\sum_i Z_i}.$$  \hspace{1cm} (13)

Based on this, we can derive k-order similarity and heterogeneity calculation methods.
6. Scheduling Strategy based on Spatio-Temporal Heterogeneity

In this section, we comprehensively consider the spatial heterogeneity between executors and the temporal heterogeneity between online executor sets and propose an index to evaluate potential harm. Based on this index, we suggest a scheduling strategy that can reduce potential harm. The scheduling strategy also maintains uncertainty, guaranteeing that our model is both learnable and nondeterministic.

**Spatial heterogeneity:** For each scheduling cycle, each executor in the online executor set has its vulnerabilities. This time, the security of the system is determined by executors’ common vulnerabilities, which are focused on by earlier mimic protection models.

**Temporal heterogeneity:** Temporal heterogeneity does not consider the duration of common vulnerabilities; it only considers the extent of common vulnerabilities all at once. If the common vulnerability of a mimic defense system is small but its common vulnerability remains the same after scheduling, then even if the probability of an attacker’s successful attack is small, once he succeeds, it will pose a long-term threat to the system and cause great losses. Furthermore, in practical scenarios, attackers are often memorable, which means that they are more likely to employ attack strategies that have worked in the past. Therefore, we also need to minimize the intersection of vulnerabilities exposed by adjacent scheduling cycles.

We use the historical scheduling information to guide the scheduling of the next scheduling cycle.

**Assumption 6.1.** The attacker has the same probability of taking an attack action at each moment. Let \( p_{att} \) denote it.

**Definition 6.1.** Let \( O(t) \) represent the set of common abnormal inputs of the online executor set at time \( t \).

\[
O(t) = V(f_{t,1}) \cdot V(f_{t,2}) \cdots \cdot V(f_{t,n}),
\]

where \( V(f_{t,i}) \) denotes the abnormal input set of the \( i \)th executor at time \( t \).

Let \( p_t \) represent the probability of the attack being successful if the attacker attacks at time \( t \), and \( p_t \) can be calculated as

\[
p_t = P(O(t)).
\]

From the previous analysis, \( p_t \) can be estimated by \( P'(O(t)) \). Let \( p_{t\rightarrow t+k} \) represent the probability that the attack at time \( t \) is still effective at time \( t+k \). If the attack launched at time \( t \) is still valid at time \( t+k \), it must be valid in every scheduling cycle between \( t \) and \( t+k \), otherwise, it will be discovered by the defense model. So \( p_{t\rightarrow t+k} \) can be calculated as

\[
E_{t\rightarrow t+k} = \frac{|O(t) \cdot \cdots \cdot O(t+k)|}{|O(t)|}.
\]

We hope that through scheduling, the attacks are as few as possible in the \( t \) scheduling period. The attack in the scheduling period \( t \) is either a successful attack launched by the attacker at time \( t \) or a successful attack launched before time \( t \) and still valid at time \( t \). The probability that the attack launched at time \( t \) succeeds and enters the system is \( p_t \), and the probability that an attack launched at time \( t-k \) succeeds and is still effective at time \( t \) is \( p_{t-k} \cdot p_{t-k\rightarrow t} \). Let \( E_t \) denote the expectation of attacks in the scheduling period \( t \). For a given scheduling policy, The calculation method of \( E_t \) is as follows,

\[
E_t = p_t + p_{t-1\rightarrow t}p_{t-1} + p_{t-2\rightarrow t}p_{t-2} + \ldots \tag{17}
\]

Let \( R(n) \) denote the remainder of \( E_t \).

\[
R(n) = p_{t-k\rightarrow t}p_{t-n} + p_{t-(n+1)\rightarrow t}p_{t-(n+1)} + \ldots \tag{18}
\]

Assuming \( E_t \) is bounded, then when \( n \) is large enough, \( R(n) \) can be arbitrarily small. Therefore, we can discard the remainder and compute only the top \( k \) terms of the \( E_t \).

Equation (17) shows the expectation of attacks in the scheduling period \( t \), but in practical application scenarios, we prefer to minimize the loss caused by the attack. Next, we modify Equation (17) to estimate the expectation of the loss caused by the attack.

**Assumption 6.2.** The destructive capabilities of the attackers are all the same. In other words, the loss caused by the attacker to the system through the attack is only related to the duration of the attack.

Let \( h(T) \) denote the loss to the system caused by the attacker’s continuous attack for \( T \) duration. Then, the expectation of the loss caused by the attack can be estimated as

\[
E(Harm)_t = p_t h(1) + p_{t-1\rightarrow t}p_{t-1}(h(2) - h(1)) + \ldots + p_{t-2\rightarrow t}p_{t-2}(h(3) - h(2)) + \ldots \tag{19}
\]

Therefore, the choice of scheduling strategy can be transformed into an optimization problem: Minimize \( E(Harm)_t \) by choosing the online executors set. We then solve this optimization problem using algorithms such as Monte Carlo simulations or heuristic methods that guarantee some randomness.

7. Analysis

7.1. Complexity Analysis

We discuss the computational complexity of our proposed algorithm separately.

**Dynamic Update Algorithm:** To perform a clustering algorithm based on fitness proportionate sharing, it is first necessary to calculate \( \frac{N(2n-N-1)}{2} \) metrics to obtain the fitness value of each candidate cluster center, where \( n \) represents the total number of the input chunk and \( N \) represents the number of candidate cluster centers. Then, at most \( N \) computations are required for fitness proportional sharing. Therefore, the total number of calculations of the clustering algorithm based on fitness proportionate sharing is \( \frac{N(2n-N+1)}{2} \).
Next, we discuss the computational complexity of the process of merging clusters. We assume that an executor saves at most $N_c$ clusters. The cluster merging process requires us to compute at most $N_c \times N_c$ metrics and at most $N_c$ cluster density updates. Therefore, the overall complexity of the clustering algorithm based on fitness proportionate sharing is $N_c^2 + N_c$.

**Scheduling Algorithm:** Calculating the $p_i$ requires $\frac{n(n-1)}{2} N_c^2$ metric calculation. Calculating $p_{\text{mut}+k}$ requires at most $kn^2 N_c^2$ metric calculations. Suppose we utilize the scheduling information of the previous $L_{ST}$ times, then calculating $E_i$ requires at most $\frac{1}{2}kn^2 (L_{ST} + 1)(nL_{ST} + n - 1)$ metric calculations. Assuming that $E_i$ needs to be calculated $K$ times in the heuristic algorithm, the total computational complexity of the scheduling algorithm is $\frac{1}{2}K n N_c^2 (L_{ST} + 1)(nL_{ST} + n - 1)$.

### 7.2. Cost analysis

The cost of our method can be mainly divided into the following four parts: the cost of heterogeneity, the cost of dynamic, the cost of redundancy, and the cost of cleaning and refactoring the executors, which we discuss in detail below.

**The cost of heterogeneity:** When deploying a mimic defense system, a large number of heterogeneous executors are required to ensure the security of the system. We assume that the cost of constructing each heterogeneous executor is the same, and denoted by $\text{cost}_f$. Then, the cost of heterogeneity can be evaluated as:

$$Cost_H = \text{cost}_f \times N_f,$$

where $N_f$ represents the number of executors in a heterogeneous executor pool.

**The cost of dynamic:** When performing mimic scheduling, replacing the executor requires a certain cost. This cost is related to the size of the online executor set, and the replacement of executors will have a certain impact on the normal work of the system, such as causing additional delays to the system. The cost of replacing the executor is linearly related to the number of the size of online executor set, and the additional loss caused by scheduling is often a constant. Therefore, the cost of executing a schedule can be evaluated as

$$Cost_D = \text{cost}_{D_1} + n \times \text{cost}_{D_2},$$

where $\text{cost}_{D_1}$ represents the cost of replacing an executor, $\text{cost}_{D_2}$ represents the loss from scheduling, $n$ represents the size of online executor set. Assuming that the scheduling period is $T$, the scheduling cost per unit time is

$$Cost_D = \frac{\text{cost}_{D_1} + n \times \text{cost}_{D_2}}{T}.$$  

**The cost of redundancy:** A reliable mimic defense system needs to run multiple executors with the same function at the same time, which brings an additional cost to the whole system. In unit time, the cost of redundancy can be evaluated as

$$Cost_R = \text{cost}_{R_0} \times n,$$

where $\text{cost}_{R_0}$ represents the cost per unit of time to run an executor.

**The cost of cleaning and refactoring the executors:** In practice, we often need to clean and refactor some fragile executors, which will bring a certain cost. We assume that the cost of cleaning and refactoring each executor is the same, and let $\text{cost}_C$ denote it. $\text{cost}_C$ is linearly related to the number of times the system cleans and refactors the executors.

$\text{Cost}_H$ is a one-time cost, and $\text{Cost}_D$, $\text{Cost}_R$ are costs per unit of time, which means that the cost of dynamic and redundancy will continue to increase over time, and the cost of refactoring and cleaning the executor depends on the actual situation. Then, the total cost $\text{Cost}_T$ can be expressed as

$$\text{Cost}_T = \text{Cost}_H + t(\text{Cost}_{R_0} + \text{Cost}_D) + \text{Cost}_C \times N_C,$$

where $t$ represents the length of time the SIDMD runs. $N_C$ stands for the number of times cleaning and operations were performed.

### 7.3. Simulation

We did random simulations to verify the reliability of SIDMD and compare SIDMD with state-of-the-art like DHR [6]. We assume that the input space is the region $[-50, 50] \times [-50, 50]$ on a two-dimensional Euclidean space and randomly generate some circular areas in this area as the vulnerabilities of each executor. We randomly generate some points within this region as inputs.

We first verify the effectiveness of the dynamic update algorithm of the SIDMD model. We randomly generate vulnerabilities for each executor and then compare them with the clusters discovered and saved by the SIDMD model. The result is shown in Figure 4. We can see from Figure 4 that the vulnerabilities discovered by the SIDMD model are very similar to the real ones.

Next, we verify that the performance of the DHR model is greatly improved after adding the dynamic update algorithm and the scheduling algorithm based on spatiotemporal heterogeneity. The settings of the relevant parameters and results can be seen in Table 1 and Table 2. (We say the DHR model added the dynamic update algorithm and the heterogeneity calculation algorithm as IDMD). We use a Monte Carlo simulation algorithm to solve the optimal scheduling policy.
(a) real vulnerabilities

(b) clusters discovered and saved by the SIDMD

Figure 4. Comparison between true vulnerabilities and discovered vulnerabilities.

### TABLE 1. MAIN PARAMETERS AND VALUES.

| Main parameters                  | Value |
|----------------------------------|-------|
| Total number of executors       | 50    |
| Number of vulnerabilities of each executor | 10    |
| Radius of the vulnerability      | 6     |
| Total number of inputs          | 100000|
| Size of the online executor set  | 5     |
| Scheduling period               | 5000  |
| $L_{ST}$                        | 2     |
| $r$                             | 2     |
| $r_0$                           | 4     |

### TABLE 2. COMPARISON OF DHR, IDMD, AND SIDMD.

| Index | DHR | IDMD | SIDMD |
|-------|-----|------|-------|
| N     | 590 | 58   | 47    |
| P     | 0.98%| 0.10%| 0.08% |
| T     | 1671226 | 156926 | 119430 |
| n     | 36  | 2    | 0     |
| P2    | 0.0610169 | 0.034483 | 0     |
| ET    | 2832.5864 | 2705.621 | 2541.064 |

In Table 2, N, P, T, P2, and ET, represent the number of successful attacks, the probability of the attacker successfully attacking, the total time of being attacked, the probability that the attacker can still attack successfully after scheduling, and the average attack time, respectively. From Table 2, we can see that the defense performance of IDMD and SIDMD is significantly better than DHR, and IDMD and SIDMD perform similarly in the number of attacks and the probability of being attacked. However, since the IDMD model only focuses on minimizing the common vulnerabilities at the same time, and thus ignores the relationship between the common vulnerabilities in adjacent scheduling cycles, the total duration of the attack is significantly longer than that of SIDMD. Therefore, SIDMD can not only reduce the probability of vulnerability exposure but also reduce the duration of the same vulnerability exposure, which is the best among the three models.

**Remark 7.1.** Since we assume that we do not know the vulnerabilities of each executor at the beginning, the IDMD and SIDMD model also uses a randomly selected scheduling method at the beginning. Until the vulnerability of each executor is updated to a certain level before starting to execute our scheduling strategy. Therefore, the number of attacks here is not counted from the beginning, but from the execution of our scheduling policy.

Finally, we study the difference in cost between SIDMD and DHR when their defensive performance is similar. After many experiments, we found that the security performance of SIDMD with 5 online executors is similar to that of DHR with 7 online executors. Our parameter settings and results are shown in Table 3 and Table 4, respectively.

### TABLE 3. PARAMETER SETTINGS OF SIDMD WITH 5 ONLINE EXECUTORS AND DHR WITH 7 ONLINE EXECUTORS.

| Main parameters                  | Value |
|----------------------------------|-------|
| Total number of executors       | 25    |
| Number of vulnerabilities of each executor | 10    |
| Radius of the vulnerability      | 6     |
| Total number of inputs          | 50000 |
| Size of the online executor set  | 7     |
| scheduling period               | 5000  |
| $SL_{(ST)}$                     | 2     |
| $Sr$s                           | 2     |
| $Sr_0$s                         | 4     |

### 7.4. Validation on real dataset

Darknet traffic classification is important for classifying real-time applications. Analyzing dark web traffic can help monitor malware ahead of time and detect
malicious activity after an attack has broken out. Our experimental data is selected from the darknet dataset: https://www.unb.ca/cic/datasets/darknet2020.html. There are about 260,000 pieces of data and 78 valid features.

We verify the effectiveness of SIDMD at a specific moment in the poisoning attack scenario. We choose one or more of StatP [11], Nopt [12], alfa [13] as the attack, and TRIM [11], Proda [12], K_LID_SVM [13] as the defense respectively. We select 2000 data sets as datasets with a class balance, among which, 1800 are used as training sets (for [13] we chose 1600 training sets, 200 validation sets), and 200 as validation sets. The poisoning rate is set to 20%, and the mixed attack is set to 10% each, and they attack without interfering with each other.

The linear model without any regularization is selected as the linear model of StatP [11] and Proda [12], the initialization of the attack selection is InvFlip, the optimization argument is x (instead of x and y), and the model is a white-box model. For defense, the maximum number of iterations γ is set to 30. Since the model used above are regression-oriented, we convert it to binary classification by setting a threshold.

SVM [13] is a soft-interval Gaussian kernel SVM, the model parameter selection is consistent with the original experimental code, C = 0.39685026299204973 GAMMA = 0.7937005259840995. The attack adopts an adaptive poisoning attack, and the defense is the default defense.

SIDMD selects 5 classifiers as its executive body, including linear model, SVM, KNN (cluster classifier), random forest (tree structure classifier), and Naive Bayes (probabilistic model classifier). Without loss of generality, both the linear classifier and the SVM classifier in DHR are the probabilistic model classifier. Without loss of generality, both the linear model of StatP and Proda, the initialization of the attack selection is InvFlip, the optimization argument is x (instead of x and y), and the model is a white-box model. For defense, the maximum number of iterations γ is set to 30. Since the model used above are regression-oriented, we convert it to binary classification by setting a threshold.

SVM [13] is a soft-interval Gaussian kernel SVM, the model parameter selection is consistent with the original experimental code, C = 0.39685026299204973 GAMMA = 0.7937005259840995. The attack adopts an adaptive poisoning attack, and the defense is the default defense.

SIDMD selects 5 classifiers as its executive body, including linear model, SVM, KNN (cluster classifier), random forest (tree structure classifier), and Naive Bayes (probabilistic model classifier). Without loss of generality, both the linear classifier and the SVM classifier in DHR are the same as above.

The experimental results we obtained are shown in Table 5. Given the different baselines of different models, the accuracy cannot be directly compared. We compare the accuracies before and after the attack to obtain the performance retention rate, as shown in Table 6. From this result, we can conclude that the defense performance of the SIDMD model is effective and robust to different attacks.

**Effectiveness:** From Table 6, SIDMD has good defensive performance against various attacks, including against different individual attacks and mixed attacks.

**Robustness:** For different types of attacks, the performance of Proda and TRIM is not stable enough: Proda resists alfa and StatP mixed attacks, the performance retention rate is 0.9595, but when resisting StatP attacks, the performance retention rate is only 0.896, but the performance retention rate is 0.9133 when defending against StatP attacks. But SIDMD can show a good and stable defense effect against different kinds of attacks.

### Table 4. The security performance of SIDMD with 5 online executors and DHR with 7 online executors.

| Num | DHR N | SID P | DHR n | SID P2 | DHR ET |
|-----|-------|-------|-------|-------|--------|
| 1   | 404   | 331   | 0.81% | 0.66% | 1273480|
| 2   | 291   | 157   | 0.58% | 0.31% | 906417  |
| 3   | 340   | 320   | 0.68% | 0.64% | 869117  |
| 4   | 315   | 255   | 0.63% | 0.51% | 793453  |
| 5   | 214   | 292   | 0.43% | 0.58% | 582114  |
| 6   | 215   | 307   | 0.43% | 0.61% | 533119  |
| 7   | 247   | 331   | 0.49% | 0.66% | 827266  |
| 8   | 129   | 331   | 0.26% | 0.66% | 331022  |
| 9   | 210   | 237   | 0.42% | 0.47% | 519501  |
| 10  | 318   | 209   | 0.64% | 0.42% | 867537  |
| Ave | 268.30| 277.00| 0.01 | 0.01 | 752502.60|

### Table 5. Multiple attacks and multiple defense test results: accuracy.

| Attack model   | Linear with Proda | Linear with TRIM | SVM with K_LID_SVM |
|----------------|-------------------|------------------|-------------------|
| StatP [11]     | 0.79              | 0.7              | 0.91              |
| Nopt [12]      | 0.815             | 0.765            | 0.89              |
| alfa [13]      | 0.83              | 0.77             | 0.91              |
| alfa & StatP   | 0.795             | 0.775            | 0.885             |

### Table 6. Multiple attacks and multiple defense test results: performance retention rate.

| Attack model   | Linear with Proda | Linear with TRIM | SIDMD |
|----------------|-------------------|------------------|-------|
| StatP          | 0.9133            | 0.8092           | 1     |
| Nopt           | 0.9422            | 0.8844           | 0.978 |
| alfa           | 0.9191            | 0.896            | 0.9725|
| alfa & StatP   | 0.9595            | 0.8902           | 1     |

Since it is difficult to know whether the data is poisoned or not in practice, we also tested the effect of defense on the classification accuracy when the data is not poisoned.

### Table 7. Deploy defense without attack performance.

| Index | Linear with Proda | Linear with TRIM | SVM with K_LID_SVM | SIDMD |
|-------|-------------------|------------------|-------------------|-------|
| Accuracy | 0.785             | 0.7              | 0.77              | 0.91  |
| Performance retention rate | 0.9075           | 0.8092           | 0.9112            | 1     |

As can be seen from Table 7, when there is no attack, only SIDMD has no decrease in the accuracy after deployment, and the rest of the defense methods have varying degrees of influence on the classification effect of the model. Therefore, to deploy defenses such as Proda and TRIM, it is necessary to know in advance when the attacker will...
attack, otherwise it will have side effects on the model. However, deploying the SIDMD model does not require prior knowledge, which manifests its advantages.

8. Related Work

The forms of attacks against cyberspace are becoming increasingly sophisticated, and the need for cybersecurity to defend against specific attacks in specific domains is becoming more prominent. Currently, most of the work is point-to-point research on attack and defense strategies, deploying one defense against one type of attack. For example, the ability of machine learning algorithms, as tools that can assist people in making decisions for large projects, to defend against various attacks determines the future of machine learning algorithms. In addition to model attacks that are effective on specific models [20], attackers can cause poisoning attacks by manipulating or maliciously injecting anomalous data during training [21]. Video recognition [22], communication networks [23], malware detection [24], edge computing [25], and domain name resolution [16] have demonstrated actual poisoning attacks.

For specific models, support vector machines [13], linear regression [17], logistic [14], generative adversarial networks [26], and Bayesian networks [27] have been studied concerning the attack and defense, but they only stay in point-to-point attack and defense. Sandamal et al [13] proposed a weighted support vector machine, an approach that significantly reduces the classification error rate by computing K-LID for feature classification in high-dimensional transform space. Wen et al [17] introduced probability estimation of clean data points in a linear regression model to reduce errors caused by poisoned datasets by optimizing the integrated model. Esmaeilpour et al [26] proposed a novel GAN architecture trained with the Sobolev integral probability metric to improve the performance of the model in terms of stability and the total number of learned patterns. However, these defense techniques can only target specific attacks of specific models and are less robust in the face of multiple attacks. Therefore, for defense research in cyberspace, upgrading from point-to-point specific defense to point-to-point active defense will be the future trend.

In recent years, Moving Target Defense (MTD) [28] has received more and more attention in the field of network security. In recent years, there have been many Defense Studies on false data injection attacks [29], [32], adversarial attacks [33], [34], and DDoS attacks [35], [36]. MTD aims to build a dynamic, heterogeneous, and uncertain target environment in cyberspace to increase the attack difficulty of attackers. This method has pioneered the defense against external attacks by changing internal structures. However, it still does not break away from the traditional thinking of additional defense, and the problem of backdoor or trap door is difficult to defend [37]. Therefore, it is necessary to introduce the redundancy theory in mimic defense to improve the overall defense performance of the system. The existing research on mimic defense mostly focuses on two aspects: heterogeneous measurement and scheduling strategy.

(1) Heterogeneity metric: For the metric of heterogeneity and similarity of an executor, it is necessary to extract the common features of endogenous security, i.e., to make a reasonable definition of the vulnerabilities of an executor so that normalized comparisons can be made between the vulnerabilities of different executors. Liu et al [38] divided the executor into n parts, each of which has several optional components, and denoted the set of vulnerabilities of the executor components by the symbol, which is extended to the executor by considering the magnitude of the common-mode vulnerabilities among the components that make up the executor. Wang et al [39] consider the different costs of common-mode vulnerabilities among different components of the executor and introduce weights in combining the common-mode vulnerabilities of each component of the executor, whose values are determined by the threat level of vulnerabilities in each layer. Zhang et al [40] calculate the heterogeneity by comparing the higher-order heterogeneity between multiple executions is defined by comparing whether vulnerabilities exist simultaneously between the “alleles” of multiple executions in turn. [38], [39], [40] all give strong assumptions on the definition of the vulnerability of the executor, and often use symbols to represent the vulnerability. In practical applications, it is often difficult for us to find out its practical significance.

(2) Scheduling strategy: most of the current studies on scheduling strategies consider the optimal defense performance at the same time, lacking the consideration of the time dimension [6], [38]-[41]. Wu et al. proposed a completely randomized scheduling strategy algorithm in [6], which generates the set of online executors by a completely randomized method. Liu et al. [38] proposed a randomized seed scheduling strategy algorithm based on the minimum similarity between online executors (RSSSA). This algorithm randomly selects an executor as the seed executor and then schedules a set of executors with minimum similarity. Wang et al. [39] proposed a scheduling strategy based on Bayesian Stackelberg game theory. In the web server scenario, this algorithm can obtain the online actuator set by computing the difference between the online and offline actuator sets to maximize the security gain on the defense side. In addition, Yang et al. [41] proposed a policy algorithm with feedback capability, which calculates the scheduling probability of an executor based on a table of historical information. They verified the defensive performance of their algorithm by designing simulated collision experiments. For the historical confidence of executors over time and the heterogeneity among executors, Zhang et al. [40] proposed an optimal scheduling algorithm. They evaluated the historical confidence of the executors by setting a sliding window, which can effectively improve the operational efficiency of the algorithm and the security of the system under non-uniform distributed network attacks.
9. Conclusion

We perform the first systematic study on a general theoretical model of cyberspace defense. We propose a way to define vulnerabilities from the input space and a new method for calculating heterogeneity. We also design an algorithm that can dynamically update vulnerabilities based on historical attack information and have better defense performance when encountering highly dynamic attacks. To improve the lack of temporal knowledge in existing scheduling algorithms, we propose a scheduling strategy based on temporal heterogeneity, SIDMD, which significantly reduces the probability of attackers detecting common vulnerabilities in defense systems. We did numerous simulations and evaluate the performance of SIDMD on a real-world application subjected to multiple attacks simultaneously and compare SIDMD with state-of-the-art. We believe that our work will inspire future research in related fields to develop more secure anthropomorphic defense systems against various unknown attacks.

References

[1] P. Chen, F. Li and J. Li, “Research on Intrusion Detection Model Based on Bagged Tree,” in IEEE Conference on Telecommunications, Optics and Computer Science (TOCS), 2021, pp. 579-582.

[2] T. Sethi and R. Mathew, “A Study on Advancement in Honeypot based Network Security Model,” in Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICTV), 2021, pp. 94-97.

[3] K. Yoshioka, Y. Hosobuchi, T. Orii and T. Matsumoto, “Vulnerability in Public Malware Sandbox Analysis Systems,” in 10th IEEE/IPSJ International Symposium on Applications and the Internet, 2010, pp. 265-268.

[4] M. Garcia, A. Bessani, I. Gashi, N. Neves and R. Obelheiro, “OS unknown attacks. SIDMD with state-of-the-art. We believe that our work will improve the performance of SIDMD on a real-world application in defense systems. We did numerous simulations and evaluate the probability of attackers detecting common vulnerabilities in defense systems. We did numerous simulations and evaluate the performance of SIDMD on a real-world application subjected to multiple attacks simultaneously and compare SIDMD with state-of-the-art. We believe that our work will inspire future research in related fields to develop more secure anthropomorphic defense systems against various unknown attacks.

References

[1] P. Chen, F. Li and J. Li, “Research on Intrusion Detection Model Based on Bagged Tree,” in IEEE Conference on Telecommunications, Optics and Computer Science (TOCS), 2021, pp. 579-582.

[2] T. Sethi and R. Mathew, “A Study on Advancement in Honeypot based Network Security Model,” in Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICTV), 2021, pp. 94-97.

[3] K. Yoshioka, Y. Hosobuchi, T. Orii and T. Matsumoto, “Vulnerability in Public Malware Sandbox Analysis Systems,” in 10th IEEE/IPSJ International Symposium on Applications and the Internet, 2010, pp. 265-268.

[4] M. Garcia, A. Bessani, I. Gashi, N. Neves and R. Obelheiro, “OS diversity for intrusion tolerance: Myth or reality?,” in IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN), 2011, pp. 383-394.

[5] M. Carvalho and R. Ford, “Moving-Target Defenses for Computer Networks,” in IEEE Security & Privacy (S&P), vol. 12, no. 2, Mar.-Apr. 2014, pp. 73-76.

[6] C. Qi, J. Wu, H. Chen, H. Yu, H. Hu and G. Cheng, “Game-Theoretic Analysis for Security of Various Software-Defined Networking (SDN) Architectures,” in IEEE 85th Vehicular Technology Conference, 2017, pp. 1-5.

[7] Q. Zhang, H. Tang, W. You and Y. Li, “A Method for Constructing Heterogeneous Entities Pool in NFV Security Architecture Based on Mimic Defense,” in IEEE 6th International Conference on Computer and Communications (ICCC), 2020, pp. 1029-1033.

[8] L. Pu, J. Wu, H. Ma, Y. Zhu and Y. Li, “MimicCloudSim: An environment for modeling and simulation of mimic cloud service,” China Communications, vol. 18, no. 1, pp. 212-221, Jan. 2021.

[9] X. Sang and Q. Li, “Mimic Defense Techniques of Edge-Computing Terminal,” in IEEE Fifth International Conference on Big Data Computing Service and Applications, 2019, pp. 247-251.

[10] Z. Lin, K. Li, H. Hou, X. Yang and H. Li, “MDFS: A mimic defense theory based architecture for distributed file system,” in IEEE International Conference on Big Data, 2017, pp. 2670-2675.

[11] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru and B. Li, “Manipulating machine learning: Poisoning attacks and countermeasures for regression learning,” in IEEE Symposium on Security and Privacy (S&P), May 2018, pp. 19-35.

[12] J. Wen, B. Z. H. Zhao, M. Xue, A. Oprea and H. Qian, “With Great Dispersion Comes Greater Resilience: Efficient Poisoning Attacks and Defenses for Linear Regression Models,” IEEE Transactions on Information Forensics and Security (TIFS), vol. 16, pp. 3709-3723, 2021.

[13] S. Weerasinghe, T. Alpcan, S. M. Erfani and C. Leckie, “Defending Support Vector Machines Against Data Poisoning Attacks,” IEEE Transactions on Information Forensics and Security (TIFS), vol. 16, pp. 2566-2578, 2021.

[14] S. Mei and X. Zhu, “Using machine teaching to identify optimal training-set attacks on machine learners,” in National Conference on Artificial Intelligence AAAI Press, 2015.

[15] J. Chen, X. Zhang, R. Zhang, C. Wang and L. Liu, “De-Pois: An Attack-Agnostic Defense against Data Poisoning Attacks,” IEEE Transactions on Information Forensics and Security (TIFS), vol. 16, pp. 3412-3425, 2021.

[16] X. Li, Z. Qu, S. Zhao, B. Tang, Z. Lu and Y. Liu, “LoMar: A Local Defense Against Poisoning Attack on Federated Learning,” IEEE Transactions on Dependable and Secure Computing (TDSC), doi: 10.1109/TDSC.2021.3135422.

[17] Z. Ma, J. Ma, Y. Miao, Y. Li and R. H. Deng, “ShieldFL: Mitigating Model Poisoning Attacks in Privacy-Preserving Federated Learning,” IEEE Transactions on Information Forensics and Security (TIFS), vol. 17, pp. 1639-1654, 2022.

[18] X. Liu, H. Li, G. Xu, Z. Chen, X. Huang and R. Lu, “Privacy-Enhanced Federated Learning Against Poisoning Adversaries,” IEEE Transactions on Dependable and Secure Computing (TIFS), vol. 16, pp. 4574-4588, 2021.

[19] X. Yan, A. Homaiifar, M. Sarkar, A. Girma, and E. W. Tunstel, “A clustering-based framework for classifying data streams,” in Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI), August 2021, pp. 3257–3263.

[20] M. Khoosrevy, K. Nakamura, Y. Hirose, N. Nitta and N. Bahaguchii, “Model Inversion Attack by Integration of Deep Generative Models: Privacy-Sensitive Face Generation From a Face Recognition System,” IEEE Transactions on Information Forensics and Security (TIFS), vol. 17, pp. 357-372, 2022.

[21] R. Schuster, T. Schuster, Y. Meri, and V. Shmatikov, “Humpy dumpty: Controlling word meanings via corpus poisoning,” in IEEE Symposium on Security and Privacy (S&P), May 2020, pp. 1295–1313.

[22] S. Xie, Y. Yan and Y. Hong, “Stealthy 3D Poisoning Attack on Video Recognition Models,” IEEE Transactions on Dependable and Secure Computing (TDSC), doi: 10.1109/TDSC.2022.3163397.

[23] O. Stan et al., “Extending Attack Graphs to Represent Cyber-Attacks in Communication Protocols and Modern IT Networks,” IEEE Transactions on Dependable and Secure Computing (TDSC), vol. 19, no. 3, pp. 1936-1954, May-June 2022.

[24] C. Li et al., “Backdoor Attack on Machine Learning Based Android Malware Detectors,” IEEE Transactions on Dependable and Secure Computing (TDSC), doi: 10.1109/TDSC.2021.3094824.

[25] J. Zhou et al., “A Differentially Private Federated Learning Model against Poisoning Attacks in Edge Computing,” IEEE Transactions on Dependable and Secure Computing (TDSC), doi: 10.1109/TDSC.2022.3168556.

[26] M. Esmaeilpour, P. Cardinal and A. L. Koerich, “Multidiscriminator Sobolev Defense-GAN Against Adversarial Attacks for End-to-End Speech Systems,” IEEE Transactions on Information Forensics and Security (TIFS), vol. 17, pp. 2044-2058, 2022.

[27] O. A. Wahab, J. Bentahar, H. Otrok and A. Mourad, “Resource-Aware Detection and Defense System against Multi-Type Attacks in the Cloud: Repeated Bayesian Stackelberg Game,” IEEE Transactions on Dependable and Secure Computing (TDSC), vol. 18, no. 2, pp. 605-622, 1 March-April 2021.
[28] M. Torquato and M. Vieira, “Moving target defense in cloud computing: A systematic mapping study,” Computers & Security, vol. 92, p. 101742, 2020.

[29] M. Higgins, F. Teng and T. Parisini, “Stealthy MTD Against Unsupervised Learning-Based Blind FDI Attacks in Power Systems,” IEEE Transactions on Information Forensics and Security (TIFS), vol. 16, pp. 1275-1287, 2021.

[30] Z. Zhang, R. Deng, D. K. Y. Yau, P. Cheng and J. Chen, “Analysis of Moving Target Defense Against False Data Injection Attacks on Power Grid,” IEEE Transactions on Information Forensics and Security (TIFS), vol. 15, pp. 2320-2335, 2020.

[31] Y. Hu, P. Zhu, P. Xun, B. Liu, W. Kang, Y. Xiong, and W. Shi, “Cpmtd: Cyber-physical moving target defense for hardening the security of power system against false data injected attack,” Computers & Security, vol. 111, p. 102465, 2021.

[32] Y. Hu, P. Xun, P. Zhu, Y. Xiong, Y. Zhu, W. Shi, and C. Hu, “Networkbased multidimensional moving target defense against false data injection attack in power system,” Computers & Security, vol. 107, p. 102283, 2021.

[33] Y. Qian, Y. Guo, Q. Shao, J. Wang, B. Wang, Z. Gu, X. Ling, and C. Wu, “Ei-mtd: Moving target defense for edge intelligence against adversarial attacks,” ACM Transactions on Privacy and Security (TOPS), vol. 25, no. 3, pp. 1–24, 2022.

[34] A. Amich and B. Eshete, “Morphence: Moving target defense against adversarial examples,” in Annual Computer Security Applications Conference, pp. 61–75, 2021.

[35] Y. Zhou, G. Cheng, S. Jiang, Y. Zhao, and Z. Chen, “Cost-effective moving target defense against ddos attacks using trilateral game and multi-objective markov decision processes,” Computers & Security, vol. 97, p. 101976, 2020.

[36] Y. Zhou, G. Cheng and S. Yu, “An SDN-Enabled Proactive Defense Framework for DDoS Mitigation in IoT Networks,” IEEE Transactions on Information Forensics and Security (TIFS), vol. 16, pp. 5366-5380, 2021.

[37] J. Wu, Cyberspace mimic defense. Springer, 2020.

[38] Q. Liu, S. Lin and Z. Gu, “Heterogeneous redundancies scheduling algorithm for mimic security defense,” J. Commun., vol. 39, pp. 188-198, Jul. 2018.

[39] X. Wang, W. Yang, W. Zhang and Z. Yang, “Research on scheduling strategy of mimic web server based on BSG,” J. Commun., vol. 39, no. 22, pp. 112-120, 2018.

[40] W. Zhang, S. Wei, L. Tian, K. Song and Z. Zhu, “Scheduling algorithm based on heterogeneity and confidence for mimic defense”, J. Web Eng., vol. 19, no. 7, pp. 971-997, Dec. 2020.

[41] L. Yang, Y. Wang and J. Zhang, “FAWA: A negative feedback dynamic scheduling algorithm for heterogeneous executor”, Comput. Sci., vol. 48, no. 8, pp. 284-290, Aug. 2021.