DOI:10.1587/transfun.2024EAP1082

Publicized:2024/11/25

Paper

A self-adaptive mimic scheduling method based on fine-grained heterogeneity

Yuli YANG[†], Jianxin SONG[†], Dan YU[†], Xiaoyan HAO[†], *and* Yongle CHEN[† a)], *Nonmembers*

SUMMARY Cyber Mimic Defense (CMD) is an active defense theory emerging in recent years, and CMD improves system robustness and security by inherent uncertainty, heterogeneity, redundancy, and other characteristics. Among them, scheduling methods, which are the key technologies of CMD, directly affect the ability of mimic systems to resist vulnerabilities and backdoor attacks. However, most of the existing scheduling methods lack a careful study of executor similarity and high-order heterogeneity. Based on this, a fine-grained heterogeneity metric method that considers high-order common vulnerabilities is proposed. Then, an adaptive scheduling method that combines actuator heterogeneity and historical confidence is proposed, and the dynamics and reliability of this scheduling method are verified by simulation experiments. Specifically, under the experimental conditions of 4 and 5 executor redundancy, the experimental experiments were compared with the CRS, TIRTS and RSMHS methods. Through 80 tests, 80 scheduling cycles and the average failure probability of the system were obtained. Experimental results show that compared with the RSMHS scheduling method, the average scheduling cycle of the HCVCS scheduling method proposed in this paper increases by 42.8% and 45.3%, and the average failure probability of the system decreases by 30.4% and 24.8%.

key words: *mimic defense; adaptive scheduling; fine-grained heterogeneity metrics; high-order heterogeneity; historical confidence*

1. Introduction

The current cyberspace is characterized by its susceptibility to attacks and the difficulty in defense, facing severe security challenges. Traditional cybersecurity technologies primarily adopt a "closing the barn door after the horse has bolted" strategy to mitigate various frequent cyber threats, employing measures such as firewalls [1], intrusion detection systems [2], and honeypots [3]. In response to the asymmetric nature of the vulnerability in cyberspace, many scholars have proposed novel concepts in network defense, including moving target defense [4,5], trusted computing [6], and customized trustworthy spaces [7].

Building upon the concept of proactive network defense and the technology of moving target defense, Academician Wu Jiangxing from the Chinese Academy of Engineering further proposes the theory of mimic defense [8]. This theory is grounded in the construction of high-availability and high-reliability non-similar redundancy, coupled with a multi-mode decision-making mechanism that does not rely on rules and features for judgment. By dynamically scheduling several functionally equivalent but structurally diverse executors, the mimic defense theory effectively defends against both known and unknown vulnerabilities as well as backdoor attacks.

Mimic defense has made significant progress at the theoretical, technical, and product levels. Examples of mimic technology products include mimic routers, mimic servers, and mimic firewalls, which enable continuous validation and iterative innovation of supporting theoretical frameworks. This design is adopted in many fields with high reliability requirements, such as railway transportation and aerospace. Additionally, mimic technology has been deeply integrated into fields such as AI and IoT. For instance, in 2020, Intel proposed the "Neuromorphic Computing" architecture [9], which uses EMIB and Foveros technologies to package multimodal heterogeneous architecture chips, thereby improving the accuracy and energy efficiency of recognizing multi-source unstructured data. Another example is the mimic defense system for vehicular networks [10], which establishes a mimic analysis engine by collecting and analyzing threat data. This system can dynamically reconfigure and combine security rules for both the in-vehicle and vehicle-server ends, generating endogenous security effects.

The introduction of various security components in mimic architecture inevitably brings some software and hardware overhead. Most heterogeneous platforms achieve software heterogeneity by selecting mature heterogeneous software or transformation scripts during the development phase, which has high operability and relatively low heterogeneity costs. In terms of hardware heterogeneity, due to the different interfaces of heterogeneous hardware, corresponding control software needs to be additionally introduced. Thus, purely hardware heterogeneity is difficult to achieve. Usually, after virtualizing heterogeneous hardware, it is put into application to expand the selection range of heterogeneous executors, reducing hardware costs to software costs.

In mimic defense, the dynamic scheduling of heterogeneous executors is of paramount importance. The scheduling strategy often determines the overall security of the mimic defense architecture. The primary function of the scheduling mechanism is to control the dynamic changes of

† The authors are with College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan 030000, China.

* This work was supported in part by the Natural Science Foundation of Shanxi Province (Grant No. 20210302124395 and 20210302123131)

a) E-mail: chenyongle@tyut.edu.cn

the system to achieve security objectives. The scheduling mechanism dynamically alters the state and behavior of the system, endowing it with diversity and unpredictability. Because the system state constantly changes, attackers find it difficult to gather sufficient information within a limited time to launch effective attacks. This dynamic variation increases the difficulty for attackers to analyze and exploit the system, thereby enhancing the system's security. The scheduling mechanism can also dynamically adjust defense strategies based on real-time monitoring data and analysis results, achieving system adaptability and self-repair capabilities. When potential threats are detected, the scheduling mechanism can immediately respond and change the system configuration to mitigate risks. Scheduling plays a core role in the mimic defense framework, and its effectiveness directly determines the success of the entire defense strategy. Through the rational design and implementation of the scheduling mechanism, the security and robustness of the system can be significantly enhanced.

Simultaneously, the heterogeneity of executors is a crucial factor that scheduling strategies must consider. The greater the heterogeneity between two executors, the more challenging it is to successfully attack both simultaneously. Existing methods for measuring heterogeneity primarily focus on common vulnerabilities among similar components in two executors, neglecting common vulnerabilities and high-order common vulnerabilities among dissimilar components. There is a lack of more detailed research on the measurement of heterogeneity. Currently, both domestic and international dynamic scheduling methods either exhibit excessive regularity or are overly random. Some scheduling strategies overly rely on feedback mechanisms, failing to meet reliability requirements.

The main contributions of this paper include three aspects:

- Introducing a heterogeneity measurement method that considers common vulnerabilities and high-order common vulnerabilities among dissimilar components of executors. This method aims to measure the heterogeneity of executor sets in a more fine-grained manner.
- Utilizing the heterogeneity measurement method from contribution 1 to calculate the high-order heterogeneity of executors. Proposing an Adaptive Scheduling Algorithm based on High-order heterogeneity and Executor Historical Confidence Score (HCVCS). In this method, the historical confidence score not only considers global confidence but also takes into account local confidence. This allows executors to adaptively switch based on historical performance and current network conditions.
- Through simulation experiments, demonstrating that the proposed HCVCS scheduling method endows the mimic system with excellent dynamism and reliability.

The remaining sections of this paper are organized as follows: Chapter 2 introduces the basic architecture of mimic defense and provides an overview of recent research on scheduling methods. Chapter 3 discusses the limitations of using second-order heterogeneity as a metric in scheduling methods and introduces a tree-based method for measuring high-order heterogeneity.Chapter 4 presents the criteria for measuring the historical confidence score of executors and introduces a scheduling method that simultaneously considers high-order heterogeneity and executor historical confidence score. Chapter 5 compares the proposed HCVCS scheduling algorithm with existing scheduling algorithms through experimental studies, validating the dynamic and reliable characteristics of the HCVCS algorithm. Chapter 6 summarizing the work presented in this paper.

2. Related Work

The fundamental architecture of mimic defense is Dynamic Heterogeneous Redundancy (DHR), as illustrated in Figure 1. It mainly consists of six components: Input Proxy, Heterogeneous Executor Set, Heterogeneous Component Set, Scheduler, Online Executor Set, and Arbiter. The Input Proxy is responsible for distributing incoming data. The principle of the Input Proxy is replication and distribution, meaning the incoming data is replicated into n copies and distributed to n heterogeneous executors with identical functionality but diverse structures. Each executor operates independently, processing input data in parallel. Subsequently, each executor consolidates its results and forwards them to the Arbiter. The Arbiter generates the final decision through a specific voting algorithm. Additionally, the Arbiter's result is fed back to the Scheduler. The Scheduler, based on the current situation, uses a specific scheduling algorithm to select a subset of executors from the heterogeneous executor set for online operation. It also cleans and restores the state and data of executors about to go offline. Each executor is composed of elements belonging to the Heterogeneous Component Set. The heterogeneity of the executor set is formed due to the different distributions of these elements among executors.The dynamism, heterogeneity, and redundancy of the mimic system introduce temporal and spatial uncertainties, making it challenging for attackers to exploit system vulnerabilities. Consequently, the system attains intrinsic security features and natural immunity.

In recent years, research on scheduling methods in mimic defense has achieved some success. Yao et al. [11] proposed the Maximum Dissimilarity (MD) algorithm and Optimal Mean Dissimilarity (OMD) algorithm. These algorithms select executor sets based on the longest dissimilarity distance and optimal mean dissimilarity, respectively. However, the distance threshold is set relatively high, leading to a lack of dynamism in executors. Yang et al. [12] introduced the Feedback Artificial Weighted Algorithm (FAWA), an artificial scheduling algorithm based on historical information. This algorithm dynamically schedules executors by considering threat information from historical records but does not address the issue of differences between executors. Liu et al. [13] measured heterogeneity by utilizing the similarity between executor components. They proposed the Random Seed Minimum Similarity algorithm (RSMS) to select an executor set with the overall minimum similarity, yet it lacks consideration for executor historical confidence, and the dynamic nature when the number of executors is limited needs further investigation.Zhang et al. [14], taking into account the complexity and heterogeneity of executors, quantified executor heterogeneity using secondary entropy. They proposed the Random Seed Scheduling Algorithm based on Maximum Heterogeneity and Web Service Quality (RSMHQ), which achieved a better balance between system security and service quality. However, the algorithm requires continuous optimization of security and service quality weights based on different environments. Wu et al. [15] introduced a Random Seed Scheduling Algorithm based on Executor Heterogeneity, Performance, and Historical Confidence (RSMHQH), achieving better performance and comprehensive metrics. Nevertheless, the selection of seed executors in this method is excessively random, providing attackers with a greater chance of successful attacks.Pu et al. [16] measured executor similarity in both time and space. They proposed a Pool Scheduling Algorithm based on Priority and Time Slice (PSPT), considering common vulnerabilities between executors. This algorithm demonstrated good dynamism and time complexity. Wei et al. [17] introduced some properties of high-order heterogeneity between executors and incorporated high-order heterogeneity into the ruling algorithm. However, they did not provide a method for calculating high-order heterogeneity.Addressing the shortcomings of the aforementioned scheduling methods, this paper first presents a measurement method for high-order heterogeneity. Based on this, a scheduling method considering high-order heterogeneity and executor historical confidence is proposed.

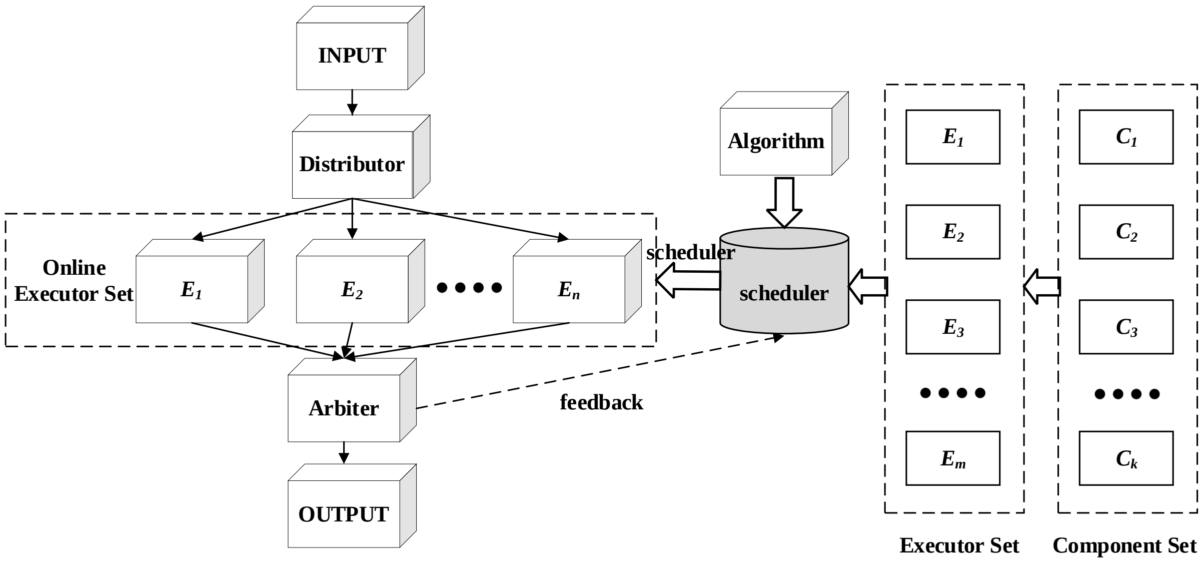

Fig. 1 DHR Architecture.

3. Fine-Grained Heterogeneity Measurement Methods

3.1 Issues with Second-Order Heterogeneity

Existing scheduling methods commonly use second-order heterogeneity [18] as the metric for assessing heterogeneity, calculating it solely based on common vulnerabilities between two executors. Relying solely on second-order heterogeneity in scheduling methods has limitations, as it neglects the consideration of high-order common vulnerabilities. Zhang et al. [19] introduced the concept of high-order common vulnerabilities:

Definition 1. High-Order Common Vulnerabilities. When different executors exhibit vulnerabilities that can achieve the same attack effects, and the number of executors satisfying this condition is denoted as 'n,' it is defined as an 'n-order common vulnerability.' Moreover, when '$n \geq 3$,' it is referred to as a 'high-order common vulnerability.'

Some of the vulnerabilities that occur in real information systems can be considered here as high-order common vulnerability. For example, the buffer overflow vulnerability CVE-2017-5123 in the operating system layer is a local privilege escalation vulnerability found in the Linux kernel, arising from an error in the kernel's error handling logic that leads to a buffer overflow. Affected systems include various Linux distributions such as Ubuntu, Debian, and CentOS, which attackers can exploit to escalate privileges and gain control over the system. The Heartbleed vulnerability is also a typical example; it is a severe flaw in the OpenSSL library that allows attackers to read protected memory, thereby stealing sensitive data. Heartbleed affects multiple layers of components, including the operating system layer, middleware layer, and database management software. Affected Linux distributions (e.g., Ubuntu, CentOS) expose the entire system to risk due to the use of affected versions of OpenSSL. Web servers (such as Apache and Nginx) that use vulnerable OpenSSL versions face risks of leaked encrypted communications, while database servers (such as MySQL and PostgreSQL) are also affected, potentially allowing attackers to read sensitive data in database memory.

Table 1 Symbol Representations.

Symbols	Definitions
M	The total number of heterogeneous executors
E_i	heterogeneous executors, $1 \leq i \leq M$
V_t	Discovered vulnerability
V	The total number of discovered vulnerabilities
S_t	The set of components in which the vulnerability V_t occurs
N	The number of components in an executor
D_{ij}	The j-th component of executor E_i, $1 \leq i \leq M$, $1 \leq j \leq N$

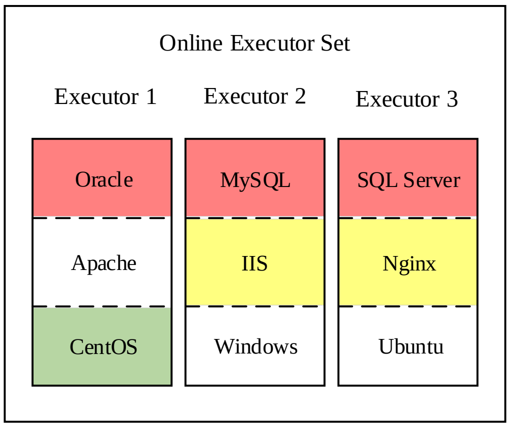

Fig. 2 High-Order Common Vulnerabilities.

As shown in Figure 2, taking a mimic web server as an example, mimic transformation can be implemented in three layers: database management software, middleware, and the operating system. Relevant symbols are explained in Table 1. Suppose the number of online executors is 3, and there are 3 types of vulnerabilities V_t (t = 1,2,3), represented by red, yellow, and blue colors at their respective locations. Different vulnerabilities can achieve different attack effects. Vulnerability V_1 appears at D_{13}, D_{23}, and D_{33}; vulnerability V_2 appears at D_{22} and D_{32}; and vulnerability V_3 only appears at D_{11}. Suppose the attacker can only discover and exploit one vulnerability within a scheduling cycle. If the attacker discovers and exploits vulnerability V_3, they can successfully attack and only attack E_1. Since executors E_2 and E_3 are not successfully attacked, the system can still output the correct result after the majority verdict. If the attacker discovers and exploits the second-order common vulnerability V_2, both executors E_2 and E_3 are compromised simultaneously. The final verdict result will be incorrect, leading to an instantaneous system breach (referred to as instantaneous attack escape). Additionally, there is a high-order common vulnerability V_1 in the executor set. If the attacker exploits vulnerability V_1 and successfully attacks all online executors, the system will be breached, and the counter-feedback control mechanism will only be triggered in the next scheduling cycle. In summary, the presence of high-order common vulnerabilities in the executor set poses a severe threat to the security of the mimic system.

3.2 Heterogeneity Measurement

Liu et al. [13] defined the relevance indicator of second-order similarity. The similarity of the n-redundancy executor set Ω^n is normalized by the sum of the similarities between all executors in the set, represented as:

$$S|_{\Omega^n} = \frac{1}{C_n^2}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} h_{ij}, \tag{1}$$

The variable h_{ij} represents the similarity between executors E_i and E_j in the n-redundancy executor set Ω^n. The detailed analysis involves the multiplication of the feature vectors corresponding to components by the feature similarity matrix, yielding the similarity between components. Subsequently, the weighted sum of the similarities among various components results in the executor-to-executor similarity, h_{ij}.

The proposed method for computing similarity only takes into account the similaity between identical components across different executors. However, considering common vulnerabilities between two executors, there might be common vulnerabilities among different components of the same or different executors. Additionally, the existence of high-order common vulnerabilities has not been considered. While matrix operations are utilized to compute component similarity, the complexity becomes excessively high when dealing with a large number of online executors.To address the limitations regarding high-order common vulnerabilities and computational complexity, this section introduces a heterogeneity measurement method based on a vulnerability M-ary tree and provides a definition for the vulnerability M-ary tree.

Definition 2. Vulnerability M-ary Tree. For a specific vulnerability V_t within the set of vulnerabilities, an M-ary tree is constructed. Taking vulnerability V_t as the root node of the tree, for any component element D_{ij} in S_t, there are two ways to add it to the tree. If no other component D_{ik} in E_i appears in the tree, component D_{ij} will be added to the tree as a child node of the root. Otherwise, D_{ij} will become a child node of the leaf node on the branch where D_{ik} is located, forming a new leaf node.Because the size of the executor set is M, according to this rule, the degree of the root node of the tree generated by this process is less than or equal to M, while the degree of other nodes is 1 or 0. It forms an M-ary tree, as illustrated in Figure 3.

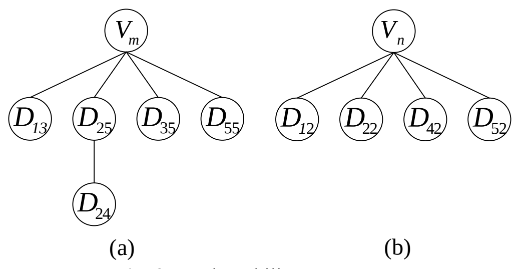

Fig. 3 Vulnerability M-ary Tree.

From Figure 3, it can be observed that the constructed Vulnerability M-ary Tree considers various scenarios. Firstly, it accounts for the situation where the same type of components from different executors share common vulnerabilities. For instance, in Figure 3(b), components D_{i2} from executors E_1, E_2, E_4, and E_5 all have the same common vulnerability V_n. Secondly, it addresses the scenario where different components from different executors exhibit common vulnerabilities. As shown in Figure 3(a), component D_{13} from executor E_1 and component D_{35} from executor E_3 share a common vulnerability Vm. Lastly, it acknowledges that different components from the same executor may also share common vulnerabilities. For example, components D_{25} and D_{24} from executor E_2 both share a common vulnerability V_m. Through the above analysis, it is evident that the constructed Vulnerability M-ary Tree allows for a more granular calculation of vulnerability positions for heterogeneity measurement. The Vulnerability M-ary Tree proposed in this paper possesses the following two properties:

Property 1. If the degree of the tree corresponding to vulnerability V_t is n, then V_t is an n-order common vulnerability.

According to the Vulnerability M-ary Tree generated by Definition 2, the degree of the tree is essentially the degree of the root node because the degree of other nodes can only be 0 or 1. If the degree of the root node of the M-ary tree corresponding to vulnerability V_t is n, it indicates that vulnerability V_t exists in n executors. According to Definition 1, V_t is considered an n-order common vulnerability.

Property 2. In the M-ary tree, if the depth of a certain leaf node D_{ij} is d_i ($d_i > 1$), it indicates that this vulnerability is a common vulnerability for d_i components within the same executor E_i.

When generating the M-ary tree for vulnerability V_n according to Definition 2, if the vulnerability appears in multiple components within the same executor, leaf nodes corresponding to these occurrences will be added. Therefore, the depth of a leaf node on a branch represents the number of components in the executor where this common vulnerability occurs.

Vulnerabilities that satisfy Property 2 will increase the attack surface for executor E_i. Considering the constructed M-ary tree and its properties, this section calculates the threat level of n-order common vulnerability V_t based on its structure. The high the order of the vulnerability, the high the system similarity, and the greater the potential threat to the system. Here, the weight function for V_t is defined as follows:

$$\theta(x,y) = \frac{1}{1+\left(e^{-x-\frac{1}{y}+\frac{n+1}{2})}\right)}, \tag{2}$$

Where x represents the order of vulnerability V_t, i.e., the degree of the M-ary tree for vulnerability V_t. y is the sum of depths of leaf nodes satisfying Property 2, i.e., $\sum d_i$ ($1 \leq i \leq M*N$), where n represents the number of online executors.

Proof. The calculation of the weight θ(x,y) for an n-order common vulnerability should conform to the changing pattern of the vulnerability threat level. Firstly, θ(x,y) must be a monotonically increasing function. Secondly, the threat level of the vulnerability increases rapidly when x $\epsilon\left[\frac{n-1}{2},\frac{n+1}{2}\right]$or y $\epsilon\left[\frac{n-1}{2},\frac{n+1}{2}\right]$. To begin, calculate the first-order partial derivative of the function θ(x,y) with respect to the variable x, assuming y∈C:

$$\frac{\partial\theta}{\partial x} = \frac{e^{\frac{n+1}{2}-x}}{\left(1+e^{\frac{n+1}{2}-x}\right)^2} > 0, \tag{3}$$

Afterwards, calculate the second-order partial derivative of θ with respect to the variable x :

$$\frac{\partial^2\theta}{\partial x^2} = \frac{e^{\frac{n+1}{2}-x}\left(e^{\frac{n+1}{2}-x}-1\right)}{\left(1+e^{\frac{n+1}{2}-x}\right)^3}, \tag{4}$$

Similarly, assuming x∈C, find the second-order partial derivative of θ with respect to the variable y :

$$\frac{\partial^2\theta}{\partial y^2} = \frac{e^{\frac{n+1}{2}-\frac{1}{y}}\left(e^{\frac{n+1}{2}-\frac{1}{y}}-1\right)}{\left(e^{\frac{n+1}{2}-\frac{1}{y}}\right)^3}, \tag{5}$$

Finally, find the zeros of the second-order partial derivative of θ with respect to the variable x :

$$\frac{\partial^2\theta}{\partial x^2} = \begin{cases} >0\,, & x\,\epsilon\left[1,\frac{n+1}{2}\right) \\ =0\,, & x=\frac{n+1}{2} \\ <0\,, & x\,\epsilon\left(\frac{n+1}{2},n\right] \end{cases}, \tag{6}$$

From equations (3) to (6), it can be concluded that $\frac{\partial\theta}{\partial x}$ >0, indicating that the weight function θ (x,y) is monotonically increasing. When x=$\frac{n+1}{2}$, the value of $\frac{\partial\theta}{\partial x}$ reaches its maximum. Thus, when x=$\frac{n+1}{2}$, the vulnerability weight increases most rapidly. This aligns with the pattern that when the vulnerability order exceeds more than half of the number of online executors, the vulnerability threat level will sharply increase, leading to instantaneous escape.When x $\epsilon\left[\frac{n+1}{2},n\right]$, the vulnerability weight gradually increases, but the rate of increase gradually decreases. This satisfies the changing pattern of the vulnerability threat level.

Definition 3. Vulnerability Binary Set: For each vulnerability V_m, there corresponds a component set S_m. Each element in the vulnerability binary set B_m represents any two components from S_m. According to the combination formula, the size of B_m is $C_{|S_m|}^2$. For executors E_i and E_j, let

$$G_k = \begin{cases} 1, & D_{jp}\ \textit{and}\ D_{iq}\ \text{appear in } B_m,\ p,q \in (1,N) \\ 0, & D_{jp}\ \text{and}\ D_{iq}\ \text{do not appear in } B_m,\ p,q \in (1,N) \end{cases}, \tag{7}$$

Traversing the vulnerability binary set of vulnerability V_m, the heterogeneity between executor E_i and E_j can be represented as:

$$\Phi_{E_iE_j} = \sum_{m=1}^{V}\left(\theta_m \sum_{k=1}^{C^2_{|S_m|}} G_k\right), \tag{8}$$

From (2) to (7), it can be observed that the larger $\Phi_{E_iE_j}$ is, the more common vulnerabilities exist between E_i and E_j, the high the likelihood of high-order common vulnerabilities, and the high the similarity between E_i and E_j . When $\Phi_{E_iE_j} = 0$, there is complete heterogeneity between E_i and E_j. Conversely, when $\Phi_{E_iE_j} = 1$, executors E_i and E_j are identical. The similarity calculation for the online executor set is as follows:

$$\Phi_E = \sqrt[M]{\sum_{i=1}^{M-1}\sum_{j=i+1}^{M}\Phi_{E_iE_j}^M}, \tag{9}$$

Finally, the pseudocode for the fine-grained heterogeneity measurement algorithm considering high-order common vulnerabilities in executors proposed in this paper is shown in Algorithm 1. According to the relationship between vulnerabilities and components input by the algorithm, the rules defined in Definition 2 and Definition 3 are used to generate the vulnerability M-ary tree and the vulnerability binary set B_i respectively. Then, the vulnerability M-ary tree is traversed to obtain the order of the vulnerability x and the sum of depths of leaf nodes that satisfy property 2 in Algorithm 2, and according to Formula 2, the vulnerability weight θ can be obtained. Finally, based on the vulnerability weight θ and the vulnerability binary set B_i, the heterogeneity of executors and executor sets is calculated.

Algorithm 1: Fine-grained Heterogeneity Measurement

```
Input: the relationship between vulnerabilities and
       components
Output: the heterogeneity of executor set $\Phi_E$ and
        executor-to-executor $\Phi_{E_iE_j}$
1  for $i = 1$ to $V$ do
       // Construct M-ary tree and vulnerability
          binary set
2      $v_i \leftarrow S_i \leftarrow \{D_{jk}\}$ j ∈ (1, M), k ∈ (1, N);
3      $Tree_i \leftarrow$ ContructTree($v_i$, $S_i$);
4      $B_i \leftarrow$ ContructSet($E$, $v_i$);
       // Determine the order and weight of each
          vulnerability
5      $currentDepth, x, y \leftarrow 0$;
6      $x, y \leftarrow$ Traversal($Tree_i$, $currentDepth$, $y$);
7      $\theta_i \leftarrow \theta_i(x, y)$;
8  end
   // Calculate the heterogeneity of executors
      set
9  $\Phi_{E_iE_j} \leftarrow \sum_{m=1}^{V}\left(\theta_m \sum_{k=1}^{C^2_{|S_m|}} G_k\right)$;
10 $\Phi_E = \sqrt[M]{\sum_{i=1}^{M-1}\sum_{j=i+1}^{M}\Phi_{E_iE_j}^M}$;
11 return $\Phi_E$, $\Phi_{E_iE_j}$;
```

Algorithm 2: Traversal

```
Input: a node of the tree $treeNode$; the depth of the
       $treeNode$ in the tree $currentDepth$; $y$
Output: $x$; $y$
1  if $currentDepth == 0$ then
2      $x \leftarrow treeNode.children.Count$;
3  end
4  if $treeNode.children.Count == 0$ then
5      if $curentDepth > 1$ then
6          $y \leftarrow y + currentDepth$;
7      end
8      return;
9  end
10 $curentDepth \leftarrow currentDepth + 1$;
11 for $i = 1$ to $treeNode.children.Count$ do
12     $Traversal(treeNode.children[i], curentDepth, y)$
13 end
14 return;
```

4. Adaptive Scheduling Method Based on Historical Confidence

4.1 Measurement Criteria for Historical Confidence

Measuring the past performance of executors to obtain their historical confidence can reflect both their historical performance and current ability to resist attacks. Currently, most research calculates the global confidence [20], representing the executor's overall historical performance. In addition, S. Gunasekaran et al. [21] proposed sliding window confidence by calculating the historical confidence within the current local time period. However, the global confidence of executors cannot fully reflect their actual attack status within the current time period, and sliding window confidence only considers the attack status within the current time period. In summary, we believe that both the global and local historical confidence of executors should be considered simultaneously, and this paper redefines the concepts and calculation methods for both.

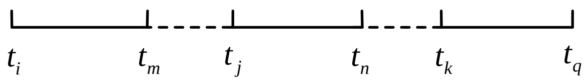

Fig. 4 Task Time Period of Executor E_i.

As shown in Figure 4, we represent the time points when executor E_i comes online as $\{t_i, t_j, t_k\}$, and the time points when E_i goes offline as $\{t_m, t_n\}$. In this paper, the historical performance of executor E_i during the period $[t_i, t_k]$ is considered as the global confidence C_{global}. The period$[t_k,\ t_q]$ represents the time span after E_i comes online at t_k, and E_i may either be replaced or continue to be online at t_q. Thus, the recent performance of executor E_i during the period $[t_k,\ t_q]$ is regarded as the local confidence C_{local}.

This section proposes a history-based adaptive scheduling method considering the online working time and the number of tasks executed by executors. The calculation methods for C_{global} and C_{local} are then elaborated. The relevant parameters are listed in Table 2.

Table 2 representation of parameters.

Symbols	Definitions
C_{global}	Global Confidence
C_{local}	Local Confidence
T*	Total System Runtime
N*	Total System Tasks
T^{E_i}	E_i's Cumulative Online Time at Present
N^{E_i}	E_i's Cumulative Tasks at Present
T^{*}_{local}	E_i's Cumulative Online Time at the End
N^{*}_{local}	Total Tasks in the System in Recent Time
T^{i*}_{local}	E_i's Cumulative Online Time at the End
N^{i*}_{local}	E_i's Cumulative Tasks in Recent Time

We use the following formula to calculate C_{global}:

$$C_{\mathrm{global}} = \frac{T^{\mathrm{E_i}}+N^{\mathrm{E_i}}}{T^{*}+N^{*}}, \tag{10}$$

For C_{global}, since the executor E_i continues to work online during $[t_k,\ t_q]$, we perform adaptive updates after each task, and the update rule should be consistent with the change in historical confidence as tasks succeed or fail. The formula is given by:

$$C_{\mathrm{local}}^{\mathrm{q}} = C_{\mathrm{local}}^{\mathrm{k}} + \frac{d(C_{\mathrm{local}})}{dt}, \tag{11}$$

Where

$$C_{\mathrm{local}}^{\mathrm{k}} = \begin{cases} 0.5 & \mathrm{E_i\ firstonline} \\ \frac{T^{\mathrm{E_i}}+N^{\mathrm{E_i}}}{T^{*}+N^{*}} & \mathrm{E_i\ onlineagain} \end{cases}, \tag{12}$$

When E_i successfully executes tasks during the $[t_k,\ t_q]$ period, the local confidence of E_i should increase slowly. Conversely, if the local confidence drops below a certain threshold due to a certain number of error outputs, E_i must be brought down to the offline cleaning threshold. If E_i successfully executes tasks during $[t_k,\ t_q]$, then

$$\frac{d(C_{\mathrm{local}})}{dt} = \frac{T_{\mathrm{local}}^{*}+N_{\mathrm{local}}^{*}-\left(T_{\mathrm{local}}^{i*}+N_{\mathrm{local}}^{i*}\right)+1}{2(T_{\mathrm{local}}^{*}+N_{\mathrm{local}}^{*})(T_{\mathrm{local}}^{*}+N_{\mathrm{local}}^{*}+1)}, \tag{13}$$

If E_i fails to execute a task at t_q (and this error occurrence is the j-th time during the period $[t_k,\ t_q]$),then

$$\frac{d(C_{\mathrm{local}})}{dt} = -\frac{(j+1)^2\left(T_{\mathrm{local}}^{i*}+N_{\mathrm{local}}^{i*}+1\right)}{T_{\mathrm{local}}^{*}+N_{\mathrm{local}}^{*}+2}, \tag{14}$$

Proof. After successfully executing a task once, let $p = T^{*}_{\mathrm{local}} + N^{*}_{\mathrm{local}}, q = T^{i^*}_{\mathrm{local}} + N^{i^*}_{\mathrm{local}}$. The growth rate of the local confidence is given by

$$\frac{d(C_{\mathrm{local}})}{dt} = \frac{p-q+1}{2p(p+1)} = \frac{1}{2p} - \frac{q}{2p(p+1)} < \frac{1}{2p}, \tag{15}$$

Similarly, when E_i fails to execute a task, the rate of change of the local confidence can be calculated as follows:

$$\frac{d(C_{\mathrm{local}})}{dt} = -\frac{(j+1)^2\left(T_{\mathrm{local}}^{i*}+N_{\mathrm{local}}^{i*}+1\right)}{T_{\mathrm{local}}^{*}+N_{\mathrm{local}}^{*}+2} > -(p+1)^2, \tag{16}$$

According to equations (13)-(16), after successfully executing a task, E_i's confidence slowly increases by approximately $\frac{1}{2(T_{\mathrm{local}}^{*}+N_{\mathrm{local}}^{*})}$, and rapidly decreases by approximately $(T_{\mathrm{local}}^{*} + N_{\mathrm{local}}^{*} + 1)^2$ after outputting an erroneous result. In other words, as the number of erroneous outputs increases, the decline in $C_{\mathrm{local}}^{E_i}$ becomes more significant. In summary, we calculate the global confidence to measure E_i's historical performance and use C_{global} as input to calculate C_{local}. Furthermore, C_{local} will adaptively adjust based on the efficiency of the current tasks, allowing for a better quantification of E_i's current ability to resist attack risks.

4.2 Scheduling Method Based on High-Order Heterogeneity and Confidence

The scheduling method needs to consider the feedback information from the arbiter module. When the global confidence or local confidence falls below a threshold, it is necessary to schedule the executor offline and promptly select a new executor online to further ensure the dynamic security of the system. To ensure the quality of system services, we assume that only one executor can be scheduled online or offline within a scheduling cycle. If two or more executors simultaneously reach the offline threshold (but not exceeding half of the online executor quantity), the executor with the lowest historical confidence will be the first to go offline. Moreover, this operation will be repeated in the next one or more scheduling cycles. If over half of the executors' historical confidence reaches the scheduling threshold, we assume that the system has suffered significant damage due to an attack. All online executors should immediately go offline for cleaning and recovery.

Assuming the online executor set is $E_{on} = \{E_1, E_2, \ldots, E_m\}$, and the executor set is E_{pool}. When selecting the initial online executor set, we tend to choose the set with the smallest overall similarity, which can be calculated by Formula (9). Through traversal to optimization, we can find the set with the smallest similarity. The rotation scheduling process is to take one executor offline and bring another executor online. When the historical confidence of $E_i \in E_{on}$ is below the threshold, it will be taken offline for cleaning and an executor $E_j \in E_{pool}$ with the smallest similarity to E_i will be selected to come online. After the rotation scheduling process, the new online executor set will be $E'_{on} = (E_{on}/E_i) \cup E_j$.

The pseudocode for the proposed HCVCS algorithm is presented in Algorithm 3 as follows.

Algorithm 3: HCVCS

```
Input: executor set $E_{pool}$; the heterogeneity of
       executor set $\Phi_E$ and $\Phi_{E_iE_j}$
Output: online executor set $E_{on}$; offline executor $E_i$;
        online executor $E_j$
// Initialize the online executors set
1  $\forall\ i,\ j\ \in\ (1,\ m)\ E_i,\ E_j\ \in\ E_{pool}$;
2  $\Phi_{E_{on}} = \sqrt[M]{\sum_{i=1}^{M-1}\sum_{j=i+1}^{M}\Phi_{E_iE_j}^{M}}$;
3  for each $\Phi'_{E_{on}}$ in $\Phi_E$ do
4      if $\Phi'_{E_{on}} < min\ \Phi_{E_{on}}$ then
5          min $\Phi_{E_{on}} \leftarrow \Phi'_{E_{on}}$;
6      end
7  end
8  return min $\Phi_{E_{on}}$;
// Select the offline executor
9  for each $E_i$ in $E_{on}$ do
10     $C_{\text{local}}^{E_i} \leftarrow C_{\text{global}}^{E_i} + \frac{\mathrm{d}(C_{\text{local}})}{\mathrm{dt}}$;
11     if $C_{\text{local}}^{E_i} < C_{th}$ then
12         $E_i$ offline;
13     end
14 end
15 return $E_i$;
// Select the online executor
16 for $E_j$ in $E_{pool}/E_{on}$, $E_i$ in $E_{on}$ do
17     min $\Phi_{E_j} \leftarrow \sqrt[M]{\sum_{i=1}^{M-1}\sum_{j=i+1}^{M}\Phi_{E_iE_j}^{M}}$;
18 end
19 return $E_j$;
```

5. Result

5.1 Experimental Environment

A web system generally consists of components such as applications, middleware, and an operating system. Here, we assume each executor includes 3 components. Considering that having multiple executors in a mimic system may increase resource consumption, most mimic systems with heterogeneous executors usually do not exceed 10. In the experiments, we chose the number of executors in the executor set to be 10, and the number of online executors was selected as 4 and 5. This allows us to compare various scheduling methods under different redundancy levels and reflect the relationship between mimic defense redundancy and dynamics, reliability. The experimental program was written in Python.

To validate the effectiveness of the proposed method, we need to first construct a mimic framework with the distribution of vulnerability positions in the components and then create attack scenarios for experimental verification. Heterogeneity is generated through simulation software. The generation principle is as follows: 10,000 vulnerabilities are randomly distributed among the components of 10 executors, with 1,000 placement points in each executor. The distribution ratio of vulnerabilities between components can be set as 1:2:7 based on the proportion of vulnerabilities in common applications, middleware, and operating systems. The generated number of i-th order common vulnerabilities is 6840, 2160, 810, 190 (i=1,2,3,4). It can be observed that as the order of vulnerabilities increases, the number of high-order common vulnerabilities decreases exponentially, with the highest not exceeding 5 orders. The distribution of high-order common vulnerabilities is generally consistent with the vulnerability distribution evaluated based on the National Vulnerability Database (NVD) [22].

5.2 Simulation Results and Analysis

This experiment includes dynamicity verification and reliability verification, which are reflected through the scheduling cycle and the system's failure probability, respectively. To reduce the uncertainty of the experiment, 80 tests were conducted to obtain 80 scheduling cycles and the average failure probability of the system. Comprehensive comparisons were made with typical completely random scheduling algorithms [23] (CRS), time-based random threshold scheduling algorithms [24] (TIRTS), and random seed scheduling algorithms based on maximum security degree and heterogeneity [25] (RSMHS). The results validate that the HCVCS algorithm possesses good dynamicity and reliability.

5.2.1 Dynamicity Verification

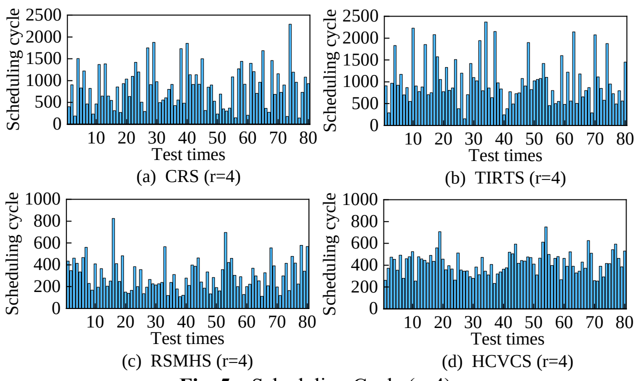

Fig. 5 Scheduling Cycle (r=4).

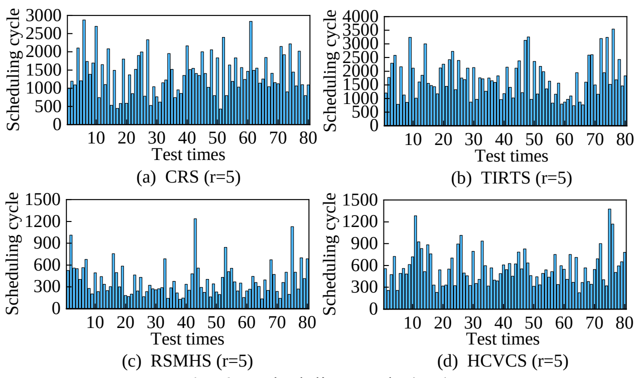

Fig. 6 Scheduling Cycle (r=5).

In the dynamicity verification, the scheduling cycle for each scheduling algorithm is obtained through multiple scheduling rounds. A scheduling cycle does not specifically refer to a time period, but rather denotes the number of times the online executor set returns to its initial state. It is independent of the positions of the executors. For example, if the initial online executor set is (E_1, E_3, E_4, E_5), and after n scheduling rounds, the online executor set becomes (E_4, E_3, E_1, E_5), then one scheduling cycle is n-1. At this point, one experiment is completed.

Table 3 Average Scheduling Cycles for 4 Algorithms.

algorithms	r = 4	r = 5
CRS[23]	855.10	1396.97
TIRTS[24]	1008.30	1742.88
RSMHS[25]	306.00	391.92
HCVCS	437.00	569.70

Figure 5 and Figure 6 show the scheduling cycles for the four scheduling algorithms with online executor redundancy of 4 and 5, respectively. The average scheduling cycles for the four algorithms are presented in Table 3. From Table 3, it can be observed that the scheduling cycles of HCVCS are greater than RSMHS algorithm but still less than CRS and TIRTS algorithms. For example, when r = 4, compared to the RSMHS scheduling algorithm, HCVCS increases the average scheduling cycles by 42.8%, while CRS and TIRTS algorithms have average scheduling cycles approximately twice that of HCVCS. This is because each executor in TIRTS and CRS is randomly selected, while HCVCS considers the heterogeneity of executors and their historical confidence levels during scheduling. Historical confidence levels often vary due to external factors. However, the scheduling cycles of TIRTS and CRS are relatively dispersed and random, whereas HCVCS exhibits more concentrated scheduling cycles. In systems where stability is a priority, HCVCS is a better choice.

5.2.2 Reliability Verification

The most intuitive way to measure the reliability of a mimic system is to assess whether the arbiter's results can tolerate attacks on certain heterogeneous executors within the mimic system. The success rate of attacks on the mimic system is the most direct indicator of its reliability. When an attacker targets a certain type of component vulnerability, it may cause the failure of executors with common vulnerabilities, leading to results different from the correct ones.

The system failure probability proposed by Zhang et al. [26] is related to the failure probability of individual executors and the heterogeneity among executors. The failure probability of executors in the experiment is generated by a normal distribution with parameters (0, 0.1). Moreover, multiple system failure probability values are obtained within each scheduling cycle, so the system failure probability corresponding to one scheduling cycle is the average of all system failure probability values obtained during that cycle.

Figure 7 shows the system failure probabilities for the four algorithms with a redundancy of 4, and Figure 8 displays the system failure probabilities with a redundancy of 5. The reliability of the CRS algorithm is relatively low because the executor selection is random, ignoring the impact of executor heterogeneity. For the RSMHS algorithm, the use of a random seed method prevents the selection of heterogeneity from reaching the global maximum. Therefore, its reliability is high than the CRS algorithm but lower than the HCVCS algorithm. The TIRTS algorithm allocates a random online time for an executor, meaning it goes offline after reaching that time and then randomly selects another executor to go online. Thus, the system average failure rate of the TIRTS algorithm is slightly lower than that of the CRS algorithm but still high than the RSMHS and HCVCS algorithms.

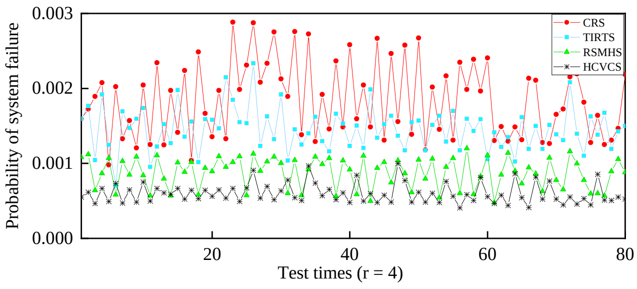

Fig. 7 System Failure Probability (r = 4)

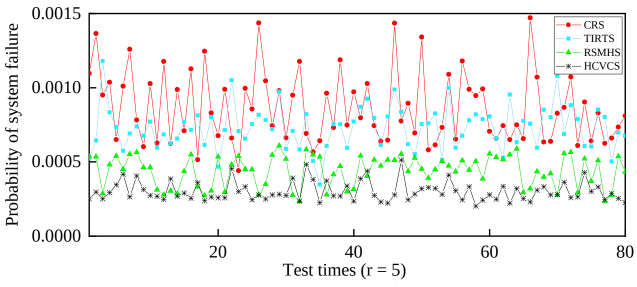

Fig. 8 System Failure Probability (r = 5)

Table 4 the average system failure probabilities.

redundancy	Algorithms			
	CRS(%)	TIRTS(%)	RSMHS(%)	HCVCS(%)
r = 4	0.1848	0.1467	0.0879	0.0612
r = 5	0.0864	0.0730	0.0440	0.0331

In Table 4, the average failure probabilities of the four algorithms are summarized. Clearly, HCVCS algorithm achieves better system reliability at the same redundancy compared to the other three algorithms. For example, when r=4, the system average failure probability of HCVCS is 30.4% lower than RSMHS. When r=5, it is 24.8% lower than RSMHS. Moreover, at both redundancy levels, the system average failure probability of HCVCS is significantly lower than CRS and TIRTS. The reason HCVCS achieves the lowest average system failure probability is that it considers the heterogeneity between executors during scheduling. When measuring the heterogeneity among executors, it calculates the degree and weight of higher-order common vulnerabilities by constructing a vulnerability M-ary tree, thereby effectively preventing the mimic system from being attacked by exploiting these higher-order common vulnerabilities. Furthermore, HCVCS takes into account the historical confidence of the executors, which adaptively adjusts according to the current task execution situation, reflecting the current capability of the executors to resist attack risks. To ensure the overall security of the mimic system at all times, executors with confidence below a threshold will be scheduled offline. As the number of experiments increases, the system failure probability correspondingly decreases.

6. Conclusions

Mimic defense technology enhances system security by introducing dynamic, heterogeneous, redundant, and negative feedback characteristics, effectively increasing the dynamic variation within the internal structure. Executor scheduling methods play a crucial role in mimic defense. This paper focuses on researching and implementing scheduling methods, achieving certain results. To address the issue of existing heterogeneity measurement methods neglecting common vulnerabilities and higher-order common vulnerabilities that arise between different types of components, a heterogeneity measurement method based on vulnerability M-ary trees is proposed. This method allows for a more granular calculation of vulnerability locations during online scheduling, helping to prevent the emergence of higher-order common vulnerabilities in the online set of executors after scheduling. To simultaneously reflect the historical performance of the executors and their current capability to resist attacks, both global and local historical confidence of the executors are considered during offline scheduling to improve the accuracy and reliability of the scheduling. Simulation experiments validate that the HCVCS scheduling method exhibits good dynamism and reliability.

References

[1] R. Oppliger, "Internet security: firewalls and beyond" Commun. ACM, vol.40, no.5, pp.92-102, 1997.

[2] R.D. P. Roberto and L. V. M. Luigi, eds., Intrusion detection systems, Springer Science & Business Media, Berlin, 2008.

[3] R. Perdisci, D. Dagon, W. Lee, P. Fogla, M. Sharif, "Misleading worm signature generators using deliberate noise injection," in Proc. 2006 IEEE Symposium on Security and Privacy, 2006, pp.15-31.

[4] J. J. Zheng and A. S. Namin, "A survey on the moving target defense strategies: an architectural perspective," Journal of Computer Science and Technology, vol.34, no.1, pp.207-233, 2019.

[5] J. H. Cho, D. P. Sharma, H. Alavizadeh, S.yoon, N. Ben-Asher, T. J. Moore, D. S. Kim, H. Lim, 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.

[6] C. Shen, D. Zhang, J. Liu, H. Ye, S. Qiu, “The strategy of TC 3.0: A revolutionary evolution in Trusted Computing,” Strategic Study of CAE, vol. 18, pp. 53-57, 2016.

[7] J. Cong, V. Sarkar, G. Reinman, A. Bui, "Customizable domain-specific computing," IEEE Design & Test of Computers, vol.28, no.2, pp.6-15, 2011.

[8] J. Wu, "Research on cyber mimic defense," Journal of Cyber Security, vol.1, no.4, pp.1-10, 2016.

[9] Y. S. Yang, Y. Kim, "Recent trend of neuromorphic computing hardware: Intel's neuromorphic system perspective," in Proc. 2020 International SoC Design Conference (ISOCC), pp. 218-219, 2020.

[10] Y. He, X. W. Liu, H. L. Ma, "Research on mimic defense system of Internet of vehicles," Journal of Computer Science and Technology, vol.6, no.3, pp.244-251, 2020.

[11] W. B. Yao, X. Z. Yang, "Design of selective algorithm for diverse software components," Journal of the Harbin Institute of Technology, 2003.

[12] L. Yang, Y. J. Wang, J. Zhang, "FAWA: A negative feedback dynamic scheduling algorithm for heterogeneous executor," Journal of Computer Science, vol.48, no.8, pp.284-290, 2021.

[13] Q. Liu, S. Lin, Z. Gu, "Heterogeneous redundancies scheduling algorithm for mimic security defense," Journal on Communications, vol.39, no.7, pp.188-198, 2018.

[14] J. Zhang, J. Pang, Z. Zhang, M. Tai, H. Zhang, G. Nie, "Executors scheduling algorithm for web server with mimic structure," Journal of Computer Engineering, vol.45, no.8, pp.14-21, 2019.

[15] Z. Wu and J. Wei, "Heterogeneous executors scheduling algorithm for mimic defense systems," in Proc. 2019 IEEE 2nd International Conference on Computer and Communication Engineering Technology, 2019, pp.279-284.

[16] L. Pu, S. Liu, R. Ding, K. Wang, "Heterogeneous executor scheduling algorithm for mimic cloud service," Journal on Communications, vol.41, no.3, pp.17-24, 2020.

[17] S. Wei, H. Zhang, Y. Su, P. Xue, L. Wen, "Majority voting algorithm based on high-order heterogeneity for mimic defense system," Journal of Computer Engineering, vol.47, no.5, pp.30-35, 2021.

[18] Z. Wu, F. Zhang, W. Guo, J. Wei, G. Xie, "A mimic arbitration optimization method based on heterogeneous degree of executors," Journal of Computer Engineering, vol.46, no.5, pp.12-18, 2020.

[19] W. Zhang, S. Wei, L. Tian, K. Song, Z. Zhu, "Scheduling algorithm based on heterogeneity and confidence for mimic defense," Journal of Web Engineering, pp.971-998, 2020.

[20] W. Guo, Z. Q. Wu, F. Zhang, and J. Wu, "Scheduling sequence control method based on sliding window in cyberspace mimic defense," IEEE Access, vol.8, no.5, pp.1517-1533, 2020.

[21] S. Gunasekaran, L. SaiRamesh, S. Sabena, K. Selvakumar, S. Ganapathy, A. Kannan, "Dynamic scheduling algorithm for reducing start time in Hadoop," in Proc. International Conference on Informatics and Analytics, 2016, pp.123.

[22] M. Garcia, A. Bessani, I. Gashi, N. Neves, R. Obelheiro, "Analysis of operating system diversity for intrusion tolerance," Software: Practice and Experience, vol. 44, no. 6, pp. 735–770, 2014.

[23] C. Qi, J. Wu, H. Hu, G. Cheng, "Dynamic-scheduling mechanism of controllers based on security policy in software-defined network," Electronics Letters, vol.52, no.23, pp.1918-1920, 2016.

[24] X. N. Sang, "Research on dynamic scheduling algorithm for mimic defense architecture," Doctor of Philosophy dissertation, Nanjing University of Science and Technology, 2020.

[25] Y. Gao, C. C. Zi, S. F. Feng, Q. Gu, "Security scheduling algorithm for web gateways based on mimicry defense theory," Journal of Chinese Computer Systems, vol.42, no.9, pp.1913-1919, 2021.

[26] S. Zhang, F. Xiao, J. H. Xu, J. Y. Li, "Determination of aviation spare parts failure rate based on similarity system theory and Bayesian theory," Electronics Optics & Control, vol.22, no.4, pp.83, 2015.

Yuli Yang was born in Yicheng, China in 1979. She obtained her M.S. degree in Computer Science and Technology from Guangxi Normal University, China, in 2007. She also received her PhD in Computer Science and Technology from Taiyuan University of Technology, China, in 2015. She is now a lecturer in College of Computer science and technology, Taiyuan University of Technology, Taiyuan, China. Her research interests are related with computer network security, cloud computing and trust management.

Jianxin Song was born in Benxi, Liaoning, China. He received his B.S. degree in IoT engineering from Taiyuan University of Technology, China, in 2022. He is currently a master's degree candidate at Taiyuan University of Technology. His interests are computer network security and IoT security.

Dan Yu was born in Taiyuan, China, in 1983. She received the B.S. degree in electronic engineering from the North University of China in 2007 and the M.S. degree in electronic engineering from the Beijing University of Posts and Telecommunications in 2013. She also received her PhD in Computer Science and Technology from Taiyuan University of Technology, China, in 2020. She is now a lecturer in College of Computer science and technology, Taiyuan University of Technology, Taiyuan, China. Her research interests are wireless sensor networks and Internet of Things.

Xiaoyan Hao was born in Xinzhou, Shanxi, China in 1970. She received a B.S. degree in Computer Science and Technology in 1992 from Shanxi University, an M.S. degree in Computer Software and Theory, and a Ph. D. in Computer Applied Technology from Taiyuan University of Technology in 2003 and 2009 respectively. She is currently a Full Associated Professor at the College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, China. Her research interests are Computational Linguistics and Information Security.

Yongle Chen was born in Weifang, Shandong, China in 1983. He received the B.S. degree and the M.S. degree, both in Computer Science, from Jilin University and Institute of Software, Chinese Academy of Science in 2007 and 2009 respectively, and the Ph.D. degree in Computer Science from University of Chinese Academy of Sciences in 2013. He is currently a Full Professor with the College of Computer science and technology, Taiyuan University of Technology, Taiyuan, China. His research interests are wireless sensor network, indoor positioning and IoT security.