IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 27, 2020, accepted October 8, 2020, date of publication October 15, 2020, date of current version October 28, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3031323

Conditional Probability Voting Algorithm Based on Heterogeneity of Mimic Defense System

SHUAI WEI[1], HUIHUA ZHANG[2], WENJIAN ZHANG[1], AND HONG YU[1]
[1]PLA Strategic Support Force Information Engineering University, Zhengzhou 450000, China
[2]Wuxi Xinwu Confidential Technology Service Center, Wuxi 214131, China

Corresponding author: Shuai Wei (ws@ndsc.com.cn)

This work was supported in part by the National Core Electronic Devices, High-end Generic Chips and Basic Software Major Projects under Grant 2017ZX01030301, and in part by the Natural Science Foundation of China for Innovative Research Groups under Grant 61521003.

ABSTRACT In recent years network attacks have been increasing rapidly, and it is difficult to defend against these attacks, especially attacks at unknown vulnerabilities or backdoors. As a novel method, Mimic defense architecture has been proposed to solve these cyberspace security problems by using heterogeneous redundant variants to perform the same task. How to choose appropriate variants and voting algorithm according to heterogeneities of these variants become the key issue of designing mimic defense architecture. Most of current researches are based on the 2-level similarity of variants, but the results are not accurate enough. This article presents an attack model based on mimic defense architecture, abstracts binary division vector and relevant indexes to describe the heterogeneity of these variants, and innovatively proposes conditional probability voting algorithm, which is different from classic majority voting algorithm. This article also analyzes the system failure probability and scalability of these voting algorithms, experiment results show that conditional probability voting algorithm is the best, both in system failure probability and scalability.

INDEX TERMS Condition probability voting algorithm, heterogeneous redundant variants, mimic defense architecture, system failure probability, scalability.

I. INTRODUCTION

With the continuous upgrading of information services, information system become more and more complex, and the amount of software and hardware code become larger and larger. Take operating system as an example, overall amount of code for Linux 2.0 is about 6MB, which increases to 28 MB for Linux 2.5, and 92MB for Linux 3.0. The latest Linux version is 5.4, total amount of code is 163 MB, which contain 25 million lines of code [1]. According to the statistics of the United States, on average, within 1000 to 1500 lines of codes, human programmers will leave a software vulnerability [2], which shows that information systems have high security risks.

Since 2010, with the emergence of Stuxnet virus [3] which infected Iran's nuclear plant system and damaged some of the centrifuges, network security issues have been widely concerned. Since then, the network attacks have expanded from traditional Internet to energy, transportation and other fields which are related to the national economy and people's livelihood. With the arrival of the Internet of things, the security situation has become more and more serious.

In order to keep cyberspace secure from network attack, some 'game-changing' technologies have been proposed, ''Moving Target Defense'' (MTD) [4]–[9] is a typical of these technologies, which make network configurations, instructions, or code locations and so on changing over time, so the attacks for these targets are becoming difficult. However, in dealing with dark features hidden in the target system or unknown attacks through the hardware/software backdoors, there still exists the problem of ineffective mechanisms [10].

How to build a safe and reliable system based on devices with unknown vulnerabilities or backdoors, Jiangxing Wu and others put forward the idea of mimic defense architecture [11]–[13], using the principle of dynamic heterogeneous redundancy to break the static accurate environment which successful network attacks usually rely on. The classic model of mimic defense architecture is shown in Fig. 1. The external input is distributed to the heterogeneous variants through

The associate editor coordinating the review of this manuscript and approving it for publication was Jun Shen.

input agent, and the heterogeneous variants perform the same task and send their results to the scheduler, the scheduler uses majority voting or alternative voting strategy to choose the correct result as final output. Then scheduler can carry out corresponding feedback control according to the results of working variants. If the number of wrong results generated by a variant is beyond threshold, then the variant is considered to have been attacked, and should be cleaned. The attacked variant will recover and be added to the work queue after cleaning. Control, system relevant parameters and system operation status can be viewed or controlled by negative feedback controls.

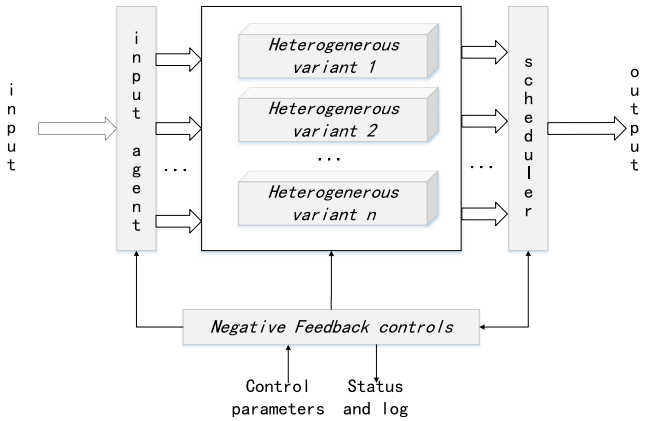


FIGURE 1. Classic model of mimic defense architecture.

The mimic defense system uses heterogeneous redundant variants to perform the same task, and compares results they generated, to make sure the results are correct. The principle is that heterogeneous variants will not or rarely generate the same error. In fact, with the development of open source and agile development technology, different heterogeneous variants inevitably have similarities. Take application program for example, there are many high-quality software in open source community, such as GitHub. After many users' testing and improvement, these software programs are relatively stable and reliable. In order to save time and reduce costs, more and more developers tend to use open source software for application development, which results in the similarity between different application programs. The same is true for hardware, such as CPU, which also uses a large number of public IP in the design process, so that there exist similarities between different hardware.

II. RELATED WORK

Software monoculture, in which many computers run identical versions of a program, is considered to be one of the basic issues that caused cyberspace insecurity [14], [15]. N-variant systems [16], [17] were introduced to solve the security problem which were firstly brought out to make system more reliable. Then they were used to defense network attacks, J. Jones use N-variant systems to defeat Denial-of-Service Attacks, such as apr pool disclosure, global overrun, heap pointer disclosure and so on [18], G. Levitin use N-variant systems to defeat coordinated attacks in cloud computing [19], B. Cox present a model for analyzing security properties of N-variant systems, define variations that can be used to detect attacks that involve referencing absolute memory addresses and executing injected code, and describe and present performance results from a prototype implementation [20].

Mimic defense system can be considered a restrict version of N-variant systems, it adopts the basic idea of running multiple variants of the same program in parallel, but there are some limitations for mimic defense system. Firstly, mimic defense system requires the variants are heterogeneous to each other, not just including applications, but also including CPU, OS, middleware and so on, secondly, mimic defense system only compare the output, no other status or middle results are taken into consideration, thirdly, there are no connection between variants to avoid coordinated attacks.

Currently most studies on N-variant systems only consider totally heterogeneous variants to defense network attacks [21]–[24], and the variants are usually referred to a relatively small program. However, for a large system which is common in mimic defense system, it is hard to realize totally heterogeneous, for example, there are limited choices of OS, only windows, Linux and VxWorks are available for common information system. For some specific field such as the control plane of Ethernet switch, only Linux and VxWorks are widely used. The combinations of module implementations are also limited, for example, uC/IP stack are sophisticate for uC/OS, but it is hard to be modified to suit for windows. So, how to build a secure system on variants that are not totally heterogeneous, became a key issue of designing mimic defense architecture, and heterogeneity was brought out to measure the degree of diversity for N-variant systems.

For mimic defense system, many studies have been done on how to choose variants for working based on heterogeneity [25]–[28], to make mimic defense system more effective in defending network attacks. The heterogeneity discussed in these researches is basically based on 2-level similarity, which is, suppose there are three variants (1, 2, 3), the 2-level similarities are referred to the similarities for 1&2, 1&3 and 2&3. The sum of similarities is lower, the system is considered to be safer. There are mainly three kinds of algorithms to choose variants: the maximum heterogeneous algorithm (MHA) [25] and the optimal mean distance algorithm (OMDA) [26], and random seeds scheduling method proposed by qinrang Liu *et al.* [27] which require the similarity of the working variants below a certain value. All these studies are based on 2-level similarities. However, no attack model has been proposed to prove their defense effect. In fact, as the number of working variants grows, 2-level similarity become less important. Jiexin Zhang measures heterogeneity based on diversity combined with the second entropy of species differences, gives a new evaluation standard of heterogeneity, and concludes that the heterogeneity will reach max when there are 3 variants in the mimic

system [28], and the heterogeneity will drop as variants number increase when the number is above 3. But only heterogeneity is taken into consideration without relevant attack model.

At present, only the heterogeneity has been introduced into the voting strategy of the mimic defense system, and consider the system is safer when the variants are more heterogeneous. B. Cox prove that the automated diversity produces variants that satisfy both the normal equivalence and detection properties against a particular attack class for N-variant systems [20]. However, no research has been done to describe the network attack model targeting the heterogeneous mimic defense system, How heterogeneities between variants takes effect in defending against network attacks, how to Quantitatively analyze system failure probability based on heterogeneity of variants and network attack model.

In this article, we analyze the mimic defense system, supposing variants are composed of module implementations, so a mimic defense system can be described as a matric of module implementations. We consider the characteristic of network attack, do some assumptions and abstract a network attacking model. We analyze the key factors that influence the defense effect of the system, extracts the module diversity, binary division vector and relevant indexes as the key indicators. We innovatively introduce conditional probability voting strategy, compare it with majority voting algorithm (MVA) [29] and maximum heterogeneous algorithm (MHA) [25] by system failure probability and scalability, prove it is best both in theory calculation and experiments. Our contributions are:

1. We abstract an attack model for mimic defense system, which provides basis for computing system failure probability of different voting algorithms.
2. We extract binary division vector and relevant indexes to measure heterogeneity of mimic defense system, which also provide basis for computing system failure probability of different voting algorithm.
3. We innovatively propose conditional probability voting algorithm (CPVA), which has lower system failure probability compared with classical MVA. We also provide the method to compute the system failure probability.
4. We prove their scalability of conditional probability voting is much better, the system failure probability does not increase as the number of working variants increase, not the same as MVA.
5. We do some experiments to prove our calculation, and prove conditional probability algorithm is best both in system failure probability and scalability.

In the rest, we begin by our problem abstraction in section 2, in which we present variant composition and network attack assumption. We present binary division vector and relevant indexes, based on these indexes and the attack assumption, we introduce CPVA, MVA and their system failure probability/scalability in section 3, In section 4 we present experiments and some evaluations. We conclude our paper in section 5.

III. PROBLEM ABSTRACTION

A. COMPOSITION OF HETEROGENEROUS VARIANTS

With the development of open source technology, more and more software and hardware adopt agile development approach to system integration. Reusing mature units can effectively reduce the cost of development, easily achieve the design goal, and improve system reliability. However, the development of code reuse technology makes it difficult in choosing heterogeneous variants. The shared units of different variants will bring homologous vulnerabilities/backdoor. The correlation between different variants can be analyzed through source code analysis, source tracing and gene analysis, etc. Generally, the more shared codes are reused, the more vulnerable mimic defense system will be.

Variants are usually composed of a series of modules, such as CPU, operating system, middleware, application, etc. Each module is composed of several modules. For example, the application can be divided into module 1, module 2,..., module M, etc. A module is the smallest unit of the variant, which is atomic (cannot be divided), and its implementations are different from each other. Then each variant can be represented by a module implementation vector $z^i = (g_1^i g_2^i \dots g_N^i)$, where N is the number of modules contained in a variant and each module may have several implementations.

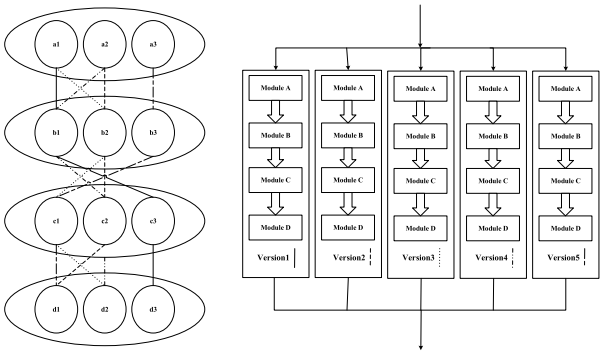


FIGURE 2. A typical mimic defense instance.

For example, Suppose there are 5 variants in the mimic defense system, as shown in Fig. 2, which is version 1 to version 5, each variant has 4 modules, each module has 3 implementations, as shown in left side of Fig. 2, there are 5 combinations of the module implementations, which are connected by different types of lines, the straight line represent version 1, the dotted line represent verson2 and so on. The mimic defense system shown in Fig.2 can be described by a matrix as shown below.

$$\begin{pmatrix} z^1 \\ z^2 \\ z^3 \\ z^4 \\ z^5 \end{pmatrix} = \begin{pmatrix} a_1 & b_1 & c_3 & d_3 \\ a_2 & b_2 & c_2 & d_1 \\ a_1 & b_2 & c_1 & d_2 \\ a_2 & b_1 & c_2 & d_2 \\ a_3 & b_3 & c_1 & d_1 \end{pmatrix}$$

B. NETWORK ATTACK ASSUMPTION

We assume that input agent, scheduler, and feedback controller shown in Fig 1 are safe from network attacks because they are realized by hardware logic. Besides, the input agent is only responsible for input data distribution. Scheduler and feedback controller are responsible for data comparison and configuration interaction, their logic is simple, and can be proved safe through formal analysis. So, failure probability of the mimic defense system is only related to the vulnerability of the variants.

Hypothesis 1: every attack against different vulnerability/backdoor will produce different abnormal output.

For the common attack methods, such as a middleware with a hidden background listening port, the communication docking with it will inevitably produce different output. Most attacks need to determine the attack effect through the output, except some unconventional attacks such as side channel attacks, but the probability is low, which is not considered in this article.

Hypothesis 2: when the high-level vulnerability / backdoor is attacked, the variants who share the vulnerability will generate the same output.

This assumption is true in most cases, such as a middleware with a hidden background listening port. But may not be true in other cases, such as buffer overflow attack. Although applications have the same buffer overflow vulnerability, due to different underlying processors, the same binary code can only be effectively executed on a specific platform (such as arm architecture processor), but not on another platform (such as X86 processors). However, attackers can also use other advanced attack methods such as branch prediction, making them to generate the same results on different platforms.

Hypothesis 3: each vulnerability / backdoor of the system have the same probability being attacked.

As described in section 1, Human programmers will leave one software vulnerability within every piece of 1000 to 1500 lines of code. Then the total number of vulnerability/backdoor of each module can be estimated by its code volume, and the attacking probability of each module can be estimated by the total number of vulnerabilities / backdoors contained in the module.

Hypothesis 4: different implementations of each module have the same attacking probability.

Due to the unpredictability of vulnerability/backdoor, the functions of the same module are the same, so the amount of code is basically the same. If the skill levels of programmers are similar, the number of vulnerabilities will be proximate. So, the error probabilities of each module's different implementations are basically the same.

Hypothesis 5: only one vulnerability / back door can be attacked at a time, and the attacked variant will be cleaned in a short time.

As the multithreading on a single core computer is actually serial at the micro-level, the attack is also serial microscopically in most cases, so only one vulnerability/backdoor can be attacked at a time. It is easy for a single variant to be breached. But the cleaning time is generally short, and the probability of another successful attack in a short time is low. Therefore, this article does not consider the case of degraded attack, that is, there is no successful attack at another variant in the cleaning process, otherwise the mimic defense system will eventually fail.

According to the above assumptions, we can assume the probability of successfully attacking each module is β_i, which should satisfy $\sum_i \sum_{\varphi_i} \beta_i = 1$, φ_i is the number of implementations for module i. The mimic defense system will generate wrong output if some shared vulnerability / backdoor is attacked and its result is voted through by scheduler. If an variant is judged to have produced an error, it will be cleaned.

C. PROBLEM TO BE SOLVED

The mimic defense system can effectively defend attacks against known or unknown vulnerability/backdoor. Take the 5-variants system in part A for example, suppose MVA is adopted, that is, the output result is considered to be correct if it is shared by majority number of variants. Then it can be seen that no matter which module implementation is attacked, at most two variants will generate the same incorrect results, and the incorrect result will not be voted through, so the system will be safe.

Voting algorithm [29]–[31] plays an important role in system failure probability of the mimic defense system. For the example shown in part A, if other voting algorithms are adopted, such as a weighted voting algorithm, that is, the results of variants 1 and 3 are preferred if they are the same. If this algorithm is adopted, when the vulnerability/backdoor in module implementation a_1 is attacked, it will cause the wrong result to be voted through and the system fails to generate correct output. So, two main problems are considered in this article:

1. When the output results of variants in the mimic defense system are inconsistent, which voting algorithm should be used to minimize the failure probability of the system? Is MVA the best one?
2. How scalability of the voting algorithm will be with the increase of variants? Can the increase of variants reduce the failure probability of the system? Which candidate variants should be selected to reduce the failure probability?

IV. CONDITIONAL PROBABILITY VOTING ALGORITHM AND MAJORITY VOTING ALGORITHM

A. BINARY DIVISION VECTORS

In order to effectively identify the degree of heterogeneity of the mimic defense system, the concepts of diversity and binary division vector are extracted to characterize the heterogeneity of modules and variants.

Module diversity φ_i : the implementation number of the module g_i, which can be calculated by formula $\left|\bigcup_i g_k^i\right|$.

Taking the module g_1 in section 2.1 as an example, its implementation set is $(a_1a_1a_1a_2a_3)^T$, then union the contents and get the set $\{a_1a_2a_3\}$, which contain 3 elements, so the diversity of module a is 3.

Binary division vector η_i^k : for a module, divide the same implementations into one group and other implementations into another group, which is called a binary division, and use a vector to represent. Assign corresponding values in the vector of the same implementations to 1 and others to 0, which is called the binary division vector of the module implementations.

The number of binary division vectors for a module is equal to the diversity of the module. For example, the implementations of g_1 in section 2.1 is $(a_1a_1a_1\ a_2a_3)^T$, then there are 3 binary division vectors for module g_1, the binary division vector of a_2 is $(00010)^T$, the binary division vector of a_3 is $(00001)^T$, the binary division vector of a_1 is $(11\ 100)^T$.

Complement binary division vector $\sim\ \eta_i^k$: which is reversing every element in the binary division vector. if adding binary division vector to its complement binary division vector, the result should be a vector whose elements are all 1.

Based on the binary division vector $(111\ 00)^T$ of module implementation a_1, reverse all the elements in it, and its complement vector will be $(0\ 001\ 1)^T$.

Isomorphic number of binary division vector λ_i^k: the number of elements whose value is equal to 1.

There are three 1 in the binary division vector of module implementation a_1, which is $(1110\ 0)^T$, then there are three a_1, and Isomorphic number of $(11100)^T$ is 3.

Property 1: Assuming that there are T executions in the mimic system, there will be at most 1 implementation whose isomorphic number is not less than $\left\lfloor \frac{T+1}{2} \right\rfloor$. If the diversity of a module is 2, there must be one implementation whose isomorphic number is not less than $\left\lfloor \frac{T+1}{2} \right\rfloor$.

B. MAJORITY VOTING ALGORITHM

MVA adopts majority rule. If the number of variants generating the same result is the largest, the corresponding result will be adopted.

1) VOTING ALGORITHM

The decision strategy of majority voting algorithm is shown as follows:

1. If there is only one result, the result is considered to be the final output result;
2. Divide the variants by their results, put variants with the same result into a group G_k. According to the hypothesis in in section 2 only one vulnerability/backdoor is attacked at a time, so there are usually 2 groups, suppose they are G_1 and G_2.
3. If $|G_1| > |G_2|$, then select the result of G_1 as the final output; otherwise, select the result of G_2 as the final output;
4. Clean the variants which have been arbitrated to be abnormal.

If the similarity of most variants in the mimic defense system was 0, the failure probability of the system will be 0. Assuming that there are T executions in the mimic system, and the Isomorphic number λ_i^k of all the binary division vectors does not exceed $\left\lfloor \frac{T+1}{2} \right\rfloor$. According to the attack assumption in section 2, the erroneous results are the same only when attacking at the variants with the same vulnerability/backdoor. Then if all the binary division vectors does not exceed $\left\lfloor \frac{T+1}{2} \right\rfloor$, once there is an attack microscopically, most variant doesn't have the vulnerability/backdoor, and will generate the correct results, then the system is absolutely safe. So, if the 2-level similarity of 3 variants' system is 0, or 3-level similarity of 5 variants' system is 0, then the failure probability of these system is 0.

2) SYSTEM FAILURE PROBABILITY

The program mainly consists of two parts. Line 2-7 saves all the modules with implementations whose Isomorphic number is no less than $\left\lfloor \frac{T+1}{2} \right\rfloor$ in stack Vg, line 9-11 sums the attacking probability corresponding to the modules in the stack to get system failure probability.

From the algorithm we can see only module that has $\left\lfloor \frac{T+1}{2} \right\rfloor$ and above implementations matters for system failure probability of T-variants mimic defense system. Other levels of similarity will have no effect.

Algorithm 1 Calculation of system failure probability of MVA

Input: module number N, variant number T,probability of module being successfully attacked β_k, module diversity φ_i, isomorphic number of binary division vector λ_i^k
Output: failure probability of MVA *msum*

```
1) Vg= NULL
2) for i = 1: N
3)     for k = 1: φ_i
4)         If( λ_i^k >= ⌊(T+1)/2⌋ )
5)             add i in Vg
6)     endfor
7) endfor
8) msum = 0
9) for each index in Ug
10)    msum = msum + β_k
11) endfor
```

3) SCALABILITY

The majority voting algorithm is relatively simple, compare the number of variants that share the same result and choose the result with largest number as the final output. However, there is a disadvantage about this algorithm, that is, increasing an variant with shared module implementation may increase the failure probability of the system, such as the

following example.

$$\begin{pmatrix} a_1 & b_1 & c_1 & d_1 \\ a_1 & b_1 & c_1 & d_1 \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{pmatrix}$$

Although the failure probability of the subset (variants at rows 3, 4, 5) is 0, but as a whole (variants at rows 1, 2, 3, 4, 5) the system failure probability is 1/3 if attacking probability of each module is equal. Because the Isomorphic number for module implementation a_1, b_1, c_1, d_1 is more than half, if the vulnerability/back door in modules a_1, b_1, c_1, d_1 is attacked, errors will be generated and the system will output incorrect results.

Although the failure probability of the system may not decrease with the increase of variants, the failure probability of the system may decrease by selecting appropriate variants. Take the variants shown in section 2.1 for example, any 3 variants from the system have a certain failure probability, but the overall 5 variants would never fail.

According to the characteristics of MVA, similar to the Buckets effect, avoiding the long and making up the short, the following conditions are generally required in order to reduce the failure probability of mimic defense system using MVA:

1. Do not add the same implementation of a module so that its Isomorphic number exceeds $\left\lfloor \frac{T+1}{2} \right\rfloor$.
2. Add different implementation of a module or balance the same implementation of a module so that its maximum implementation is less than $\left\lfloor \frac{T+1}{2} \right\rfloor$.

C. CONDITIONAL PROBABILITY VOTING ALGORITHM

The basic idea of CPVA is when the results generated by variants are different, we should analyze exactly which modules will cause current groping of results, remove the impossible modules that may cause the grouping, and find modules group with less error probability, adopt the corresponding result as the final output. Because this algorithm determines the output according to the results grouping, it is called conditional probability voting algorithm.

1) VOTING ALGORITHM

The decision strategy of CPVA is described as follows:

1. If there is only one result, take the result as the output result;
2. The variants which generated the same results are divided into one group G_K, generally there are only two groups, assumed as G_1 and G_2;
3. Calculation β_{G_1} and β_{G_2}, $\beta_{G_1} = \sum_k \beta_k$, *if* $k \in \bigcap_{i \subset G_1} (z^i - \bigcup_{j \subset G_2} z^j)$, $\beta_{G_2} = \sum_k \beta_k$, *if* $k \in \bigcap_{i \subset G_2} (z^i - \bigcup_{j \subset G_1} z^j)$
4. If $\beta_{G_1} > \beta_{G_2}$, the result of G_2 shall be used, otherwise, the result of G_1 shall be used;
5. Clean the abnormal variants which have generated wrong result.

CPVA groups variants according to their results, analyzes the modules grouping situation, remove those modules that are not able to cause the grouping situation, and select the set of modules that have low probability. According to the assumption in section 2, in each case only one vulnerability/backdoor could be attacked, so the final result of the system can be divided into G_1 and G_2. We do elimination and comparison mainly in three steps:

1. If there is the same implementation of one module in both G_1 and G_2, the error cannot be caused by attacking vulnerability/backdoor in this module. So the same module implementation in the G_1 and G_2 need to be removed, then we can get two eliminated sets G_1' and G_2', that is, $z^i - \bigcup_{j \subset G_2} z^j$ and $z^i - \bigcup_{j \subset G_1} z^j$.
2. If there are multiple implementations of one module in G_1' or G_2', the error cannot be caused by vulnerability/backdoor in these different module implementations, or there will be different results in G_1' or G_2', we use intersection to eliminate different module implementations, we can get two eliminated sets G_1'' and G_2'', that is, $\bigcap_{i \subset G_1} (z^i - \bigcup_{j \subset G_2} z^j)$ and $\bigcap_{i \subset G_2} (z^i - \bigcup_{j \subset G_1} z^j)$,
3. Now we get the shared module implementation, G_1'' and G_2'', which could cause this kind of grouping situation, then accumulate probabilities of modules contained in G_1'' and G_2'', and select the result with lower probability as final output, so the overall system failure probability will be relatively lower.

2) SYSTEM FAILURE PROBABILITY

The program is mainly divided into two parts, 1-8 lines union all the binary division vector whose corresponding isomorphic implementations are not less than $\left\lfloor \frac{T+1}{2} \right\rfloor$, save the union result in a stack *Gu*. 9-33 lines take out every element *vctorl* in the stack *Gu*, compare them with all binary division vectors in the system, save sequence number of the module who contain one implementation whose binary division vector is equal to *vctorl* in G_L, save sequence number of the module who contain one implementation whose binary division vector is complemental to *vctorl* in *Gs*, and then compare sums of attacking probabilities for the modules in G_L and *Gs*, accumulate the relatively lower probability, and finally get system failure probability.

From the description of the algorithm, it is easy to see that the failure probability of CPVA is smaller than that of MVA. Because when all the failure probabilities in G_L is not greater than those in *Gs*, the failure probability of CPVA is equivalent to the majority voting algorithm. Otherwise CPVA can get a lower failure probability.

It is easy to see that when the binary division vector is $(1\,1\cdots10)^T$, that is, Isomorphic number is N-1, there must exist another binary division vector $(00\cdots01)^T$. But con-

Algorithm 2 Calculation of system failure probability of CPVA

Input: module number N, variant number T, probability of module being successfully attacked β_{k}, module diversity φ_i, binary division vector η_i^k, isomorphic number of binary division vector λ_i^k

Output: system failure probability of CPVA *csum*

1) Gu= NULL
2) for $i = 1$: N
3) for $k = 1$: φ_i
4) if($\lambda_i^k >= \left\lfloor \frac{T+1}{2} \right\rfloor$)
5) $Gu = \mathrm{union}\ (Gu, \eta_i^k)$
6) endif
7) endfor
8) endfor
9) csum = 0
10) $G_L = G_S =$ NULL
11) for each vctorl in Gu
12) for i = 1: N
13) for k = 1: φ_i
14) if($vctorl == \eta_i^k$)
15) add i in G_L
16) else if ($\sim vctorl == \eta_i^k$)
17) add i in G_S
18) endif
19) endfor
20) endfor
21) $lsum = ssum = 0$
22) for each k in G_L
23) $lsum = lsum + \beta_{\mathrm{k}}$
24) endfor
25) for each k in G_S
26) $ssum = ssum + \beta_{\mathrm{k}}$
27) endfor
28) if $lsum < ssum$
29) $csum = csum + lsum$
30) else
31) $csum = csum + ssum$
32) endif
33) endfor

versely, when the partition vector is $(00\cdots01)^T$, there may exist multiple binary division vectors, such as $(10\cdots00)^T$, $(01\cdots00)^T$, and so on. So, the number of binary division vector $(00\cdots01)^T$ must be more than binary division vector $(11\cdots10)^T$ in the system. In the mimic defense system composed of 3 heterogeneous variants, the binary division vectors in G_L are always chosen, then CPVA is equivalent to MVP.

D. SCALABILITY

CPVA can effectively perform better scalability than MVA, such as the example of 5 variants in section 3.2. If the system adopts CPVA, the same grouping which makes MVP wrong, that is, one grouping of variants 1, 2 and 3, and another grouping of variants 4 and 5, Since there is no shared module implementations for variants 4 and 5, the conditional probability voting algorithm will choose results of variant 4 and 5 as the final output, so system failure probability is 0.

In general, it can be proved that system failure probability will not increase with increasing variants in the system, that is, the failure probability of any variants set is no less than that of its extended set. Because if binary division vectors are equal, their subsets must be equal. So, the number of shared binary division vectors in G_L and G_S must be more than that of the extended set. System failure probability comes from shared binary division vector, so system failure probability of the extended set must be no greater than system failure probability of the original set.

Although CPVA has good scalability, increasing the number of variants will not increase the system failure probability, but it does not always reduce system failure probability. To reduce the failure probability of heterogeneous variants, one of the following conditions should be satisfied:

1. Variant with different implementation of the module whose diversity number is 1 in the original system.
2. Variant with different implementation of the module in the group (G_L or G_S) with lower probability when the diversity number is more than 1 in the original system.

V. EXPERIMENTS AND ANALYSIS

Our experiment assume that every variant has N modules, implementations of each module are randomly generated within a certain range of M. M for each module is different to distinguish different modules, such as [101,110], [201,220] and so on. It is assumed that attacking probability for every module implementation is the same, so system failure probability of different voting algorithm is easy to compute. Experimental program is written with MATLAB. Three kinds of voting algorithms are tested, the maximum heterogeneous algorithm (MHA) [14], MVA as shown in section 3.2, CPVA as shown in section 3.3.

Suppose an 5 variants instance is generated as below when N=3 and M=3, take the variants at line 1, line2 and line 3 as a 3-variants system, there are 101, 201, 202, 301, 302 module implementations, so the whole vulnerabilities is 5, according to MHA algorithm which only take 2-degree similarity into consideration, which is 2 (for variants 1 and 2), 2(for variants 2 and 3),1(for variants 1 and 3), so the accumulated similarity is 5, and the failure probability is 5/5, but if adopt the algorithm described in section IV, system failure could be caused by 101, 201 and 302, so the failure probability of MVA and CPVA is 3/5. Take all the variants as a 5-variants system, there are 101, 102, 103, 201, 202, 203, 301, 302, 303 module implementations, so the whole vulnerabilities is 9, according to MHA algorithm which only take 2-degree similarity into consideration, which is 2 (for variants 1 and 2), 2(for variants 2 and 3),1(for variants 1 and 3), 1 (for variants 1 and 4), 0(for variants 2 and 4),0(for variants 3 and

4), 1(for variants 1 and 5), 1 (for variants 2 and 5), 0(for variants 3 and 5),0(for variants 4and 5), so the accumulated similarity is 8, so system failure probability is 8/9, but if adopt the MVA, the failure probability could be caused by 101, 201, so the failure probability of MVA is 2/8. if adopt the CPVA, the failure probability could be caused by no module implementations, so the failure probability of CPVA is 0.

$$\begin{pmatrix} 101 & 201 & 301 \\ 101 & 201 & 302 \\ 101 & 202 & 302 \\ 102 & 203 & 301 \\ 103 & 201 & 303 \end{pmatrix}$$

For mimic defense system, if the number of variants -N in the system increases, the system cost, power consumption, etc. would also increase. So, during the realization of mimic defense system, the number of variants would be relatively small, because of that we limit the number of variants within 10 for system failure probability test and scalability test.

A. 3-VARIANTS EXPERIMENT

We choose 10 and 100 modules, 5 and 10 module implementations range for tests, so, (N, M) combinations are (10,5), (100,5), (10,10), (100,10). We randomly generate a 10-variant set, and it contains 120 3-variant subsets, all the subsets are evaluated. The results of MRA are shown with black $*$ sign, results of MVA are shown with red O sign, and results of CPVA are shown with blue $+$ sign.

It is shown in Fig 3-6 that system failure probability of MVA completely coincides with system failure probability of CPVA, which is consistent with the conclusion in section III. In the case of 3 variants, CPVA is equivalent to MVA. It also can be seen that when N=10, the failure probability of MRA is basically the same as that of MVA, but when N=100, t the failure probability of MRA is far better than that of MVA. Because when the number of modules is large, there is a great probability that the same implementation will be shared by

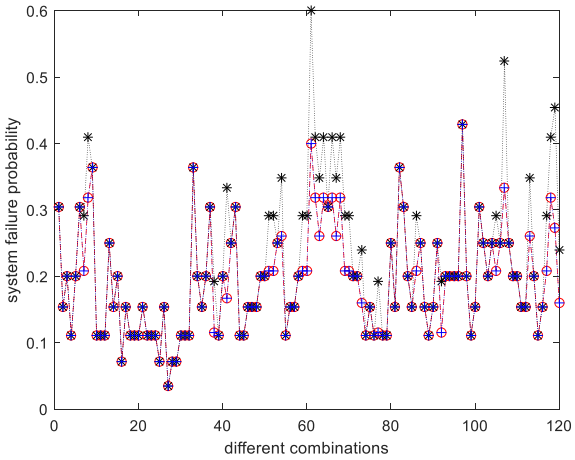


FIGURE 3. System failure probiblities Of MHA, MVA and CPVA when N=10 M=5.

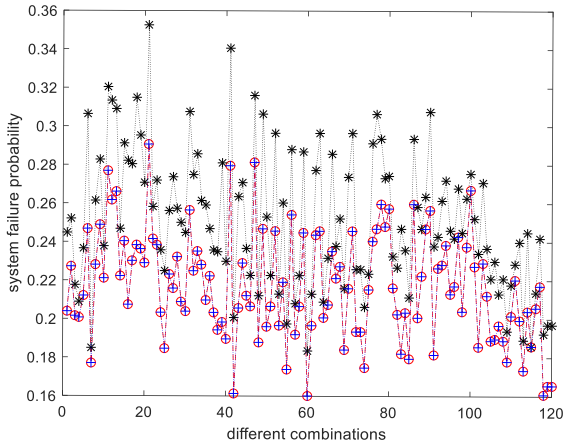


FIGURE 4. System failure probabilities of MHA, MVA and CPVA when N=100 M=5.

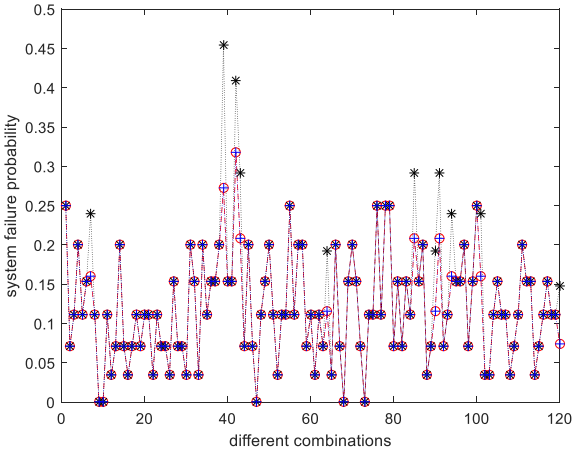


FIGURE 5. System failure probabilities of MHA, MVA and CPVA when N=10 M=10.

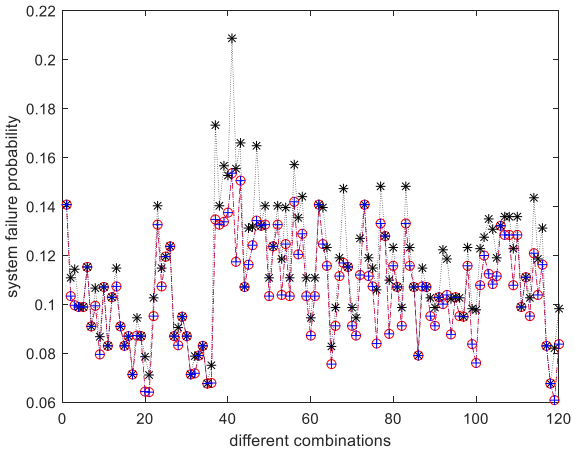


FIGURE 6. System failure probabilities of MHA, MVA and CPVA when N=100 M=10.

3 variants. In this case, MRA doesn't take 3-level similarity into consideration, so it will result in inexact evaluation. When M increases, we can see the difference between MRA and MVA/CPVA become smaller, because with the increase of module implementations range, the probability that the

same implementation of a module being shared by 3 variants become lower.

B. 5-VARIANTS EXPERIMENT

We choose 10 and 100 modules, 10 and 20 module implementations range for tests, that is, (N, M) sets are (10,10), (100,10), (10,20) and (100,20). We randomly generate a 9-variant set, and it contains 126 5-variant subsets, all the subsets are evaluated. The results of MRA are shown with black $*$ sign, results of MVA are shown with red O sign, and results of CPVA are shown with blue $+$ sign, which is shown Fig 7-10.

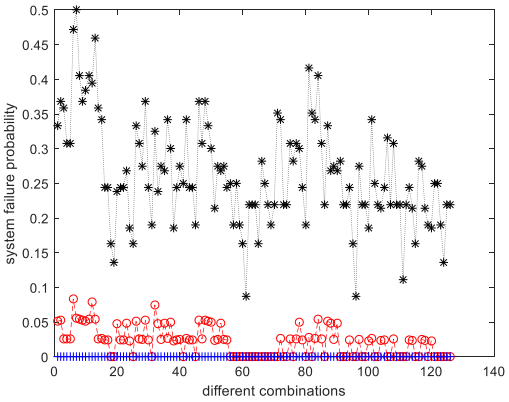


FIGURE 7. System failure probabilities of MHA, MVA and CPVA when N=10 M=10.

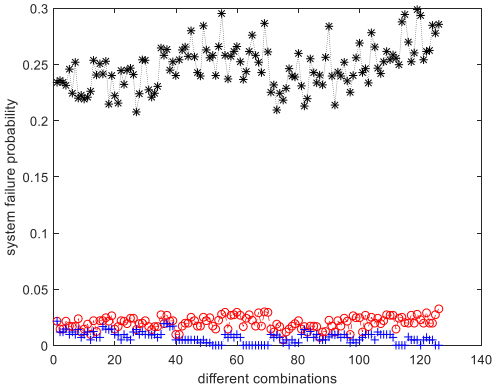


FIGURE 8. System failure probabilities of MHA, MVA and CPVA when N=100 M=10.

Apparently different from the phenomenon shown in 3-excutors system, the system failure probability of MVA is remarkably lower than that of MRA. Because in 5-excutor system, 3-level and above similarities are determining factors for system failure probability, but MRA only takes 2-level similarity into consideration. So the result is remarkably wrong. And it can be predicted that the error will be larger with the increase of variants. It is shown in these four examples that system failure probability of CPCA is different from system failure probability of MVA. It performs lower and better when M become larger or N become smaller. Because with the increase of modules or decrease of module implementations range, the probability that 3-level and above

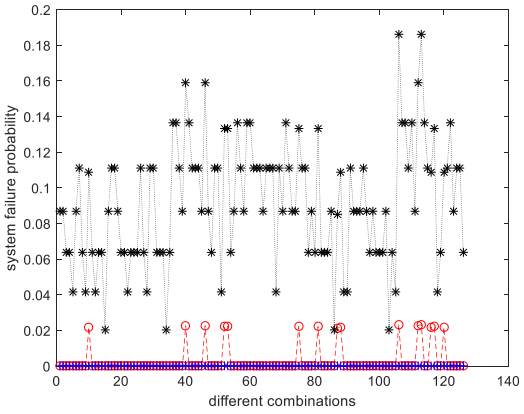


FIGURE 9. System failure probabilities of MHA, MVA and CPVA when N=10 M=20.

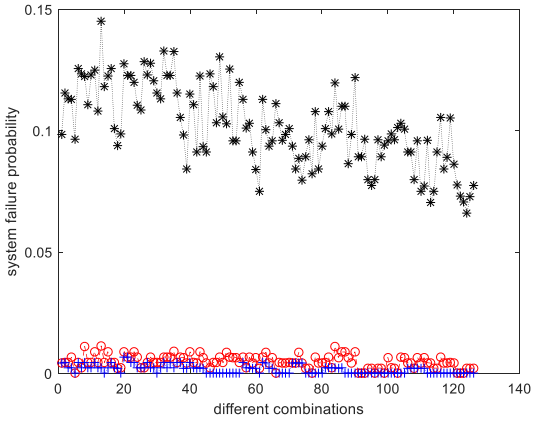


FIGURE 10. System failure probabilities of MHA, MVA and CPVA when N=100 M=10.

shared module implementation become bigger, the imbalance between GL and GS also increase, therefore CPCA will benefit more from the imbalance.

C. SCALABILITY EXPERIMENT

First 3 random variants are created as the basic mimic defense system, and then randomly created variants are added gradually to the existing system, calculate the failure probability of MVA and CPVA, and then analyze the scalability of the algorithm. The results are shown in Fig 11 and Fig 12, Lines

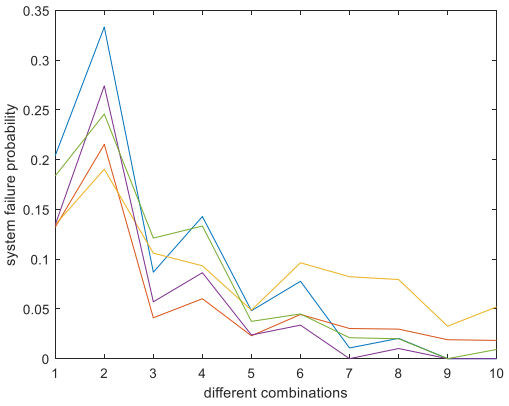


FIGURE 11. System failure probabilities of MHA with variants increase.

of different colors in the figure represent different random tests using the same method. It can be seen that the failure probability of MVP will decrease with the increase of variants in general, but it is not strictly decrease, and it will randomly jitter. While he failure probability of CPVA will strictly decline both totally and partly without jitters, which is consistent with the analysis in section 3.

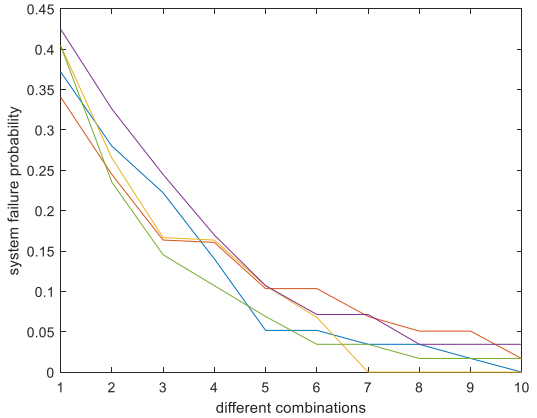


FIGURE 12. System failure probabilities of CPCA with variants increase.

In the large number voting strategy, it can be seen that the system failure probability of even number (2k) actua-tors are significantly higher than that of 2k-1 variants, so generally the odd number of variants will be adopted in the MVA, but the CPVA does not have this problem. The system failure probability of even number variants 2k is not greater than that of 2k-1 variants, and in some cases failure probability will significantly decline.

D. PERFORMANCE ANALYSIS

Mimic defense system adopts heterogeneous variants to execute the same task in parallel, which increase cost. The system collects all the variants' output and judge which is correct, which add delay of the system, and throughput may also be affected, depending on the performance of the scheduler. We realize the mimic defense system for Control panel of Ethernet Switch which is shown in Fig 13, and the scheduler

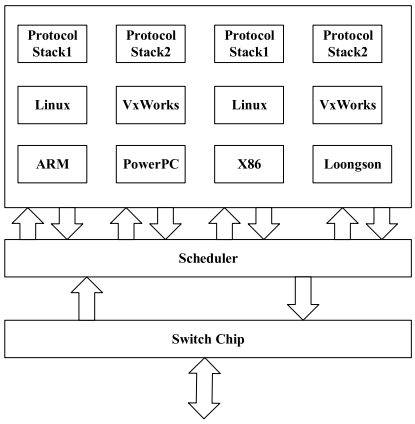


FIGURE 13. Mimic defense architecture for control panel of ethernet switch.

are realized by FPGA, which add 1ms delay for the system in average, and the system throughput is about 10Gbps.

VI. CONCLUSION

Based on the principle and architecture of mimic defense system, this article analyzes the voting algorithms of the scheduler, including majority voting algorithm and conditional probability voting algorithm, summarizes the advantages and disadvantages of various voting algo-rithms, and does corresponding experiments. Experimental results show that conditional probability voting algorithm can improves the security and reliability of the system greatly.

However, the assumption of the attack model in this article is relatively strict. There are some cases in which the variants with the same vulnerability generate different results and with different vulnerabilities generate the same results. The probabilities of different implementations of the same module being attacked may be different. Therefore, the algorithm can be extended to be suitable for more examples.

ACKNOWLEDGMENT

The authors would like to thank Prof. Jiangxing Wu, an academician of Chinese Academy of Engineering, who has proposed the main idea of mimic defense system.

REFERENCES

[1] The Linux Kernel Orgnization. (2020). *The Linux Kernel Archives*. [Online]. Available: https://cdn.kernel.org/pub/linux/kernel/

[2] X. Wang, "Research on the Semantic Annotation of Software Vulnerability Source Codes," M.S. thesis, Dept. Softw. Eng., Northwestern Polytech. Univ., Xi'an, China, 2018.

[3] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Secur. Privacy Mag.*, vol. 9, no. 3, pp. 49–51, May 2011.

[4] S. W. Boyd, G. S. Kc, M. E. Locasto, A. D. Keromytis, and V. Prevelakis, "On the general applicability of instruction-set randomization," *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 3, pp. 255–270, Jul. 2010.

[5] A. Nguyen-Tuong, D. Evans, J. C. Knight, B. Cox, and J. W. Davidson, "Security through redundant data diversity," in *Proc. IEEE Int. Conf. Dependable Syst. Netw. With FTCS DCC (DSN)*, Jun. 2008, PP. 187–196.

[6] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 371–386, May 2011.

[7] R. Zhuang, S. A. DeLoach, and X. Ou, "Towards a theory of moving target defense," in *Proc. 1st ACM Workshop Moving Target Defense (MTD)*, 2014, pp. 31–40.

[8] R. Zhuang, A. G. Bardas, and S. A. DeLoach, "A theory of cyber attacks: A step towards analyzing MTD systems," in*Proc. 2nd ACM Workshop Moving Target Defense*, 2015, pp. 11–20.

[9] J. B. Hong and D. S. Kim, "Assessing the effectiveness of moving target defenses using security models," *IEEE Trans. Dependable Secure Comput.*, vol. 13, no. 2, pp. 163–177, Apr. 2016.

[10] J. Wu. *Cyberspace Mimic Defense-Generalized Robust Control and Endogenous Security*. Berlin, Germany: Springer, 2020, pp. 15–205.

[11] J. Wu, "Research on Cyber mimic defense," *J. Cyber Secur.*, vol. 1, no. 4, pp. 1–10, Oct. 2016.

[12] H. Hu, J. Wu, Z. Wang, and G. Cheng, "Mimic defense: A designed-in cybersecurity defense framework," *IET Inf. Secur.*, vol. 12, no. 3, pp. 226–237, May 2018.

[13] S. Wei, H. Yu, and Z. Y. Gu, "Architecture of mimic security processor for industry control system," *J. Cyber Secur.*, vol. 2, no. 1, pp. 1–12, 2017.

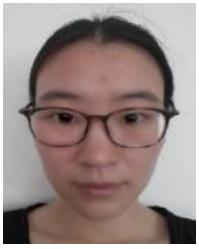
[14] M. Co, B. Dutertre, I. Mason, N. Shankar, S. Forrest, J. W. Davidson, J. D. Hiser, J. C. Knight, A. Nguyen-Tuong, W. Weimer, J. Burket, G. L. Frazier, and T. M. Frazier, "Double helix and RAVEN: A system for cyber fault tolerance and recovery," in *Proc. 11th Annu. Cyber Inf. Secur. Res. Conf. (CISRC)*, New York, NY, USA, 2016, p. 17, doi: 10.1145/2897795.2897805.

[15] D. Geer, C. P. Pfleeger, and B. Schneier, "Cyberinsecurity: The cost of monopoly," Comput. Commun. Ind. Assoc., Washington, DC, USA, Tech. Rep., Sep. 2003. [Online]. Available: http://www.ccianet.org/papers/cyberinsecurity.pdf
[16] J. Magee and T. Maibaum, "Towards specification, modelling and analysis of fault tolerance in self managed systems," in *Proc. Int. Workshop Self-Adaptation Self-Manag. Syst. (SEAMS)*, New York, NY, USA, 2006, pp. 30–36.
[17] E. Yuan and S. Malek, "A taxonomy and survey of self-protecting software systems," in *Proc. 7th Int. Symp. Softw. Eng. Adapt. Self-Manag. Syst. (SEAMS)*, Piscataway, NJ, USA, Jun. 2012, pp. 109–118.
[18] J. Jones, J. D. Hiser, J. W. Davidson, and S. Forrest, "Defeating Denial-of-Service attacks in a self-managing N-Variant system," in *Proc. IEEE/ACM 14th Int. Symp. Softw. Eng. Adapt. Self-Manag. Syst. (SEAMS)*, May 2019, pp. 126–138.
[19] G. Levitin, L. Xing, and Y. Xiang, "Co-residence data theft attacks on N-Version programming-based cloud services with task cancelation," *IEEE Trans. Syst., Man, Cybern. Syst.*, early access, Jun. 30, 2020, doi: 10.1109/TSMC.2020.3002930.
[20] B. Cox, D. Evans, and A. Filipi, "N-variant systems: A secretless framework for security through diversity," in *Proc. 15th Conf. USENIX Secur. Symp.*, Berkeley, CA, USA, 2006, pp. 1–16.
[21] S. Volckaert, B. De Sutter, and T. De Baets, "GHUMVEE: Efficient, effective, and flexible replication," in *Foundations and Practice of Security*, J. Garcia-Alfaro, F. Cuppens, N. CuppensBoulahia, A. Miri, N. Tawbi, Eds. Berlin, Germany: Springer, 2013, pp. 261–277.
[22] B. D. Rodes and A. Nguyen-Tuong, "Defense against stack-based attacks using speculative stack layout transformation," in *Runtime Verification*, S. Qadeer S. Tasiran, Eds. Berlin, Germany: Springer, 2013, pp. 308–313.
[23] E. D. Berger and B. G. Zorn, "DieHard: Probabilistic memory safety for unsafe languages," in *Proc. ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI)*, New York, NY, USA, 2006, pp. 158–168, doi: 10.1145/1133981.1134000.
[24] A. Nguyen-Tuong, D. Evans, J. C. Knight, B. Cox, and J. W. Davidson, "Security through redundant data diversity," in *Proc. IEEE Int. Conf. Dependable Syst. Netw. With FTCS DCC (DSN)*, 2008, pp. 187–196.
[25] C. Shen, S. Chen, and C. Wu, "Adaptive mimic defensive controller framework based on reputation and dissimilarity," *J. Commun.*, vol. 39, no. S2, pp. 173–180,2018.
[26] W. B. Yao and X. Z. Yang, "Design of selective algorithm for diverse software modules," *J. Harbin Inst. Technol.*, vol. 35, no. 3, pp. 261–264, 2003.
[27] Q. Liu, S. Lin, and Z. Gu, "Heterogeneous redundancies scheduling algorithm for mimic security defense," *J. Commun.*, vol. 39, no. 7, pp. 192–202, 2018.
[28] J. Zhang, J. Pang, and Z. Zhang, "Quantification method for heterogeneity on Web server with mimic construction," *Ruan Jian Xue Bao/J. Softw.*, vol. 31, no. 2, pp. 564–577, 2020.
[29] B. Parhami, "Voting algorithms," *IEEE Trans. Rel.*, vol. 43, no. 4, pp. 617–629, Dec. 1994.
[30] Z. Tong and R. Y. Kain, "Vote assignments in weighted voting mechanisms," *IEEE Trans. Comput.*, vol. 40, no. 5, pp. 664–667, May 1991.
[31] N. Jamali and C. Sammut, "Majority voting: Material classification by tactile sensing using surface texture," *IEEE Trans. Robot.*, vol. 27, no. 3, pp. 508–521, Jun. 2011.

SHUAI WEI was born in Dengzhou, Henan, China, in 1984. He received the B.S. degree in computer science and technology, the M.S. degree in software engineering, and the Ph.D. degree in high performance computing and parallel compiling in 2005, 2008, and 2012, respectively. He is currently a Research Assistant with NDSC. He has authored over 40 applied patents and published articles. His research interests include high-performance computing, cyber security, and machine learning.

HUIHUA ZHANG was born in Taizhou, Jiangsu, China, in 1983. He received the B.S. degree in computer science and technology, in 2005, and the M.S. degree in software engineering, in 2008. He is currently a Research Assistant with NDSC. He has authored over 40 applied patents and published articles. His research interests include high-performance computing, cyber security, and machine learning.

WENJIAN ZHANG was born in Shangqiu, Henan, China, in 1987. He received the B.S. degree in communication engineering, in 2010, and the M.S. degree in communication and information systems, in 2013. He is currently a Research Assistant with the Wuxi Confidential Technology Service Center. He has authored over ten applied patents and published articles. His research interests include high-performance computing, cyber security, and machine learning.



HONG YU was born in Ziyang, Sichuan, China, in 1988. She received the B.S. degree in computer science and technology, in 2011, and the M.S. degree in software engineering, in 2013. She is currently a Research Assistant with NDSC. She has authored over 20 applied patents and published articles. Her research interests include high-performance computing, cyber security, and machine learning.