Topology-Aware Software Assignment in Networked Systems

Junchao Wang, Hao Liu, Zhenwu Liu, Zheng Zhang, Jianmin Pang
State key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, China
wangjunchao11@126.com

Abstract. Cyberspace mimic defense has been proven to be a revolutionary defense technology that “changes the rules of the game” to ensure the security of cyberspace. However, the software diversity inherent in mimic defense technology may increase the difficulty in managing software executable binaries, especially when updating or debugging a software. In this paper, we study the problem of software assignment in a networked system to minimise the number of binaries generated by mimic compilers. Mimic compilers can help to generate functional equivalent software executables which exhibits diverse characteristics such as binary size etc. Theoretically the software assignment problem is equal to the traditional graph coloring problem. To guarantee network severity, we apply a Welsh-Powell-based Software Assignment (WPSA) algorithm to determine the number of binaries needed and the assignment of these binaries to hosts in a network. We conduct experiments on real world network topologies. Experimental results show that our algorithm can effectively reduce the number of binaries needed in networked systems.

1. Introduction
Cyber space security is becoming increasingly important in recent years. The number of hosts, smart phones etc. increases each year and most of them have access to Internet. Beyond the convenience brought by the Internet, cyber space security becomes a problem facing all the hosts and smart phones accessing the Internet. Typical cyber space security problems involve information leakage, network attack, swarm infection etc, which may cause tremendous disaster to economics and social life. The security problems are enabled through different attacking methods, e.g. social engineering, XSS attack, DDoS attack, memory attack etc. Most of these attacks are asymmetrical. For instance, a software may have some memory vulnerabilities. An attacker can construct explicit exploits utilizing these vulnerabilities and achieve their purpose such as controlling a computer. Practice has proved that the key cause of cyberspace security problems is the existence of known or unknown vulnerabilities and backdoors in the system [1]. According to the statistics collected by the National Information Security Vulnerability Sharing Platform, the proportion of software application vulnerabilities reached around 59% in the total number of general software and hardware vulnerabilities [2]. Therefore, most of the cyberspace security issues happened at the software level.

Worm attack is still one of the most widely spread networked attacking approaches. For instance, WannaCry is a famous worm attack happened on 12 May, 2017 which caused severe disasters to economic, energy and medical facilities. Later on, more advanced malicious ransomware named Petya or NotPetya attacked massive targets on a global scale, and many countries suffered severely. The worms are spread in a networked system where the hosts need to install the same software. For instance, network-accessible file sharing systems need to communicate with each other with certain protocols [3]. Thus, the installed software should have the same functionality. However, in existing
research [3-4], the communication between two hosts with the same binaries is defective, and the worm attack can easily spread from one host to another.

For known and unknown vulnerabilities and backdoor threats in cyberspace, mimic defense techniques can change the presentation of vulnerabilities or backdoors, block most response chains for vulnerabilities or backdoor attacks, and increase the difficulty of exploiting or backdoor exploits [5]. After a transformation to a software (e.g., inserting NOP instructions), the software may crash or even execute without affecting by the attack [6]. Moreover, software diversity has been proven to be a promising approach for preventing the spreading of worms [7].

However, in practical networking systems, it is extremely difficult to exploit software diversification and mimic defense techniques. On the one hand, in order to optimize the security of the network system, we can generate and allocate different binary files for each host. However, the diverse binaries on each host can be hard to manage and incur extra hidden cost. First and foremost, the users need to compile their source codes for many times to generate different binaries. When the scale of a software is large, the compiling process can be very slow. Generating different binaries can make the compiling time even longer. Secondly, different binaries need extra storage and careful management. An extreme situation is that for each host we can assign a different binary executable. But storing these binaries can take quite a lot of storage. Thirdly, when there are problems or bugs in some binaries, it can be very difficult and time-consuming to debug these binaries. On the other hand, the management of a large scale of hosts becomes quite difficult. If the software installed on each host in the system has the same function but different structures, although the system runtime exception can be found to the greatest extent, the generation, management, operation and maintenance of the heterogeneous software set will become the biggest difficulty facing the software manager.

Therefore, regarding the issue above, this paper attempts to minimize the executable file by analyzing the communication characteristics, and avoids the defective communication between the hosts, while maximizing the effect of the mimic defense security gain.

2. Background

2.1. Cyberspace Mimic Defense

Based on diversified technologies, the mimic defense technology adopts a “Dynamic Heterogeneous Redundancy” system architecture, which enables the system to have an endogenous security mechanism through dynamics, heterogeneity and redundancy, thereby improving system security in cyberspace. In the environment of uncertain software and hardware credibility, it has opened up a new way to solve the problem of “self-controllable, secure and credible” software and hardware components.

In the process of mimic defense technology research, the software diversity can enhance the uncertainty of software structure representation under functional equivalence conditions, combined with multi-mode ruling under non-cooperating conditions, making it difficult for attackers to form a coordinated attack escape in the mimic boundary, and ensure the security and reliability of the system software [8]. Therefore, the problem of software diversity implementation and the optimization of heterogeneous binary file deployment in the system are all key issues in the research of mimic defense technology.

2.2. Software Diversity

Software diversity can be used to better enable system security. For instance, Stijn et al. apply a Disjoint Code Layouts (DCL) technique to effectively cope with the Return Oriented Programming (ROP) attacks in multi-variant execution environment [6]. Simon et al. studied the software diversity for multi-tier web applications and took Wordpress as a use case. They proved that multi-tier software diversification is an effective way to cope with cyber threat [9]. Unitary software which exhibits same characteristics can expose precise knowledge for a wide range of attacks. Thus, Larsen proposed a probabilistic protection to increase the attackers’ attack entropy. They summarise different software
diversification approaches for different types of attacks and analyse their effectiveness [10]. Benoit et al. comprehensively reviewed research progress on software diversity since 2000. They summarised the existing software diversity approaches towards different objectives [11] such as security or fault tolerance. Hiltunen et al. propose a Cactus approach which monitors and adapts an application’s components during runtime to achieve survivability [12]. Garcia et al. evaluated the operating system’s security against security bugs of the NIST National Vulnerability Database (NVD) and proved that diversity indeed help to improve the system’s security [13]. Gorbenko et al. propose a multi-level software diversity approach which leverages the diversity of cloud’s off-the-shelf components such as operating system, web server, database management system etc [14]. Their proposed approaches can reduce system exposure to vulnerabilities. In summary, software diversity has been proven to be an effective way to deal with cyberspace diversity.

2.3. Software Assignment
Huang et al. studied the software assignment problem in different views [4]. They propose the definition of “Common Vulnerability Graph (CVG)” which is actually a sub-graph with the same color of the network topology. Their objective is to minimise the maximum severity of CVGs in the communication network topology. However, in their proposed approach, it is difficult to evaluate the vulnerabilities of a software variant over another. Moreover, they took the number of software to be installed on a host as constraint, which is not always the case in real world cases. In [3], Adam et al. try to assign a set of software to nodes on a communication network to minimise the number of neighbours with the same software package. They also assume that the number of software to be assigned is pre-defined. In this paper we don’t take such constraints. To the best of our knowledge, no existing work before this paper solve the software assignment problem from the perspective of mimic defense technologies and benchmark the solution in real world topologies.

3. Problem Statement
We use $G=<V, E>$ to represent a communication network. Each node $v \in V$ represents a host. There are $n$ vertices in total in the network, namely $|V| = n$. Each edge $e(v_i, v_j) \in E$ represents a communication link between two hosts $v_i$ and $v_j$. In this paper we assume that the graph is a connected graph. It means that $\forall v_i, v_j \in V, i \neq j$, there exists a path from $v_i$ to $v_j$. Since in this paper we consider the communication network as undirected, the path between $v_i$ and $v_j$ is also undirected. This assumption corresponds to the worm attack in real world since the worm affection is also undirected. A set of binaries $\{b_1, \cdots, b_n\}$ can be assigned to hosts in the network, denoting deployment of a binary executable on a host. All the binaries are functional equivalent. It means that if two different executables are given the same input, the output are the same.

**Definition 1.** We use $A(v_i) = b_j$ to denote assignment of host $v_i$ with binary executable $b_j$.

**Definition 2.** In the context of this paper, we introduce a new concept “acceptable assignment” to differentiate whether a worm attack can spread in a communication network. An assignment $A(V)$ represents an assignment to the hosts.

$\forall v_i, v_j \in V, i \neq j$, if $A(v_i) \neq A(v_j)$ and $v_i$ is directly adjacent to $v_j$, we call such an assignment $A$ as an acceptable assignment.

Fig. 1 shows an example of an acceptable and unacceptable assignment. We use different colors to differentiate the binary executables assigned to each host. We can see that in Fig. 1b all the hosts are assigned with different colors, meaning that all the hosts are assigned with different binary executables. In this case, the worm attack cannot be initiated. If an attack succeeds in one host $v_i$, the attack is difficult to affect another host $v_j$ that has a communication link to $v_i$. This is because the attacker has to carefully and explicitly arrange the memories that a software utilise. Even in the worst case $v_j$ may corrupt but will not be affected and work as a zombie host. This corruption can be recovered in the mimic execution environment [15]. In this paper we will not go into too much detail of this but mainly focus on the software assignment problem. The only difference in Fig. 1a is that there exists a
defective edge between \( h_1 \) and \( h_3 \). A defective edge means that \( h_1 \) and \( h_3 \) are assigned with the same binary executables. An attack succeeding in \( h_1 \) can naturally affect \( h_3 \) because of the same memory layout.

![Fig. 1: Examples of an acceptable and unacceptable assignment](image)

As described above, the software assignment problem can be equal to the classical “graph coloring” problem if we use different colors to represent different executables. In this case, any two adjacent hosts are forbidden to be assigned with a same binary. The objective in this paper is to minimise the number of different binaries needed to without any two adjacent hosts assigned with the same color.

4. Software Assignment Algorithm

Graph coloring problem has proven to be an NP-Complete problem. Thus, no efficient algorithm can find the minimum number of colors for coloring a graph in polynomial time [16]. The minimum number of colors needed is also called the chromatic number of a graph. In this paper we mainly introduce two algorithms: a simple greedy software assignment algorithm and the classical Welsh-Powell algorithm [17].

4.1. A greedy software assignment algorithm

To avoid the defective edges in a communication network, an intuitive idea is to assign each host with a different binary executable. Surely in this way any two adjacent hosts will never be assigned with the same binary. But this may increase the number of different binaries needed. Generally the different binaries can be generated with different versions. However, most software developers prefer the latest version which is more stable and having less vulnerabilities. The difference can also be enabled through compiler-based transformations. Even with different optimisation levels in compilers, the generated binaries can also be different. In this paper we use such greedy software assignment algorithm as baseline for the evaluating effectiveness of the proposed Welsh-Powell based solution.

4.2. Welsh-Powell Algorithm

Pseudo code of the Welsh-Powell algorithm is shown in Algorithm 1. The Welsh-Powell algorithm is a strategy based on maximum priority, which is colored in descending order of vertex degrees. According to Welsh Powell method, the first step is to arrange the nodes in graph \( G \) in descending order of node’s degree. Then a node \( v_i \) with maximum degree which is not assigned is chosen. A color is assigned to \( v_i \). Others nodes which is not adjacent to \( v_i \) is assigned with the same color as \( v_i \). Such procedure is repeated until all nodes are finished [18].
Algorithm 1 Welsh-Powell Algorithm

1: procedure HSSA(G,R)
2:  Sort all vertices in V in descending order based on their degree
3:  while there exists non colored vertices do
4:     Assign a non colored vertex $v_1$ in ordered V with a new color $c$
5:     for all $v \in V - \{v_1\}$ do
6:         if no adjacent vertex to $v$ has color $c$ then
7:             assign $c$ to $v$

4.3. Example
In this paper, we use the Welsh-Powell algorithm to color the undirected connected graph with 8 vertices shown in Figure 3. We use the Graphviz to visualise the original graph shown in Figure 2. From Figure 3 we can see that the graph needs at least three colors to guarantee any adjacent nodes having different colors. In Figure 2, when applying the greedy heuristic solution described above, eight different binaries are needed. However, we can reduce five binaries by analysing the communication between the hosts.

![Fig. 2: 8-nodes undirected graph before coloring](image1)

![Fig. 3: Example of coloring using the Welsh-Powell algorithm](image2)

5. Experiments
In this paper we mainly evaluate how many different binaries are needed under different network topologies by comparing the results of a greedy solution and Welsh-Powell algorithm.

5.1. Data Sets
In this paper we use real world network topologies in Topology Zoo [19] to do evaluation. For the graph with parsing exceptions with NetworkX [20](the exceptions can be caused by duplicate nodes etc.), we remove these graphs from our dataset. We also remove the graphs which is not a connected graph. Thus, we eventually evaluate 80 graphs in total.
5.2. Experimental Results

Figure 4 shows results on graphs with filenames starting with ‘A’ in Topology Zoo. “WP results” refers to the minimum number of executables given by the Welsh-Powell algorithm; “Maximum Degree” denotes the maximum node degree in each graph; “Greedy” represents the results given by the proposed greedy solution which is obviously equal to the vertex number in each graph. From the results we can see that the number of executables needed are reduced significantly with the Welsh-Powell solution.

![Fig. 4: Results of graphs with filenames starting with ‘A’](image)

Figure 5 shows results given by percentage reduced with the 80 network topologies. The vertical axis is represented in the unit of 100 percent. The graphs are sorted based on their scale. From the results we can see that with the increase on the size of the graph, the reduction becomes more clear, reaching to 14.5 times reduction at most. The number of executables needed are reduced to 5.69 times on average.

![Fig. 5: Percentage reduced in the 80 network topologies](image)

6. Conclusion and Future Works

Exploiting an attack is a complex and elaborate process by carefully crafting the attacking payload. An attack that is effective for a binary without a transformation may not take effect on another binary with a transformation. In worst cases, the attack can lead to a software crash rather than taking control of it. In this paper we apply the Welsh-Powell algorithm to minimize the number of binaries needed for a given communication networked topology. We also benchmark the effectiveness of our proposed solution on real world network topologies. Experimental results show that our approach can effectively reduce the burden of software management problem in mimic environment.

The scope of this paper is restricted to a worm-like attack. In our future works we will consider more different types of attacks. We will also further evaluate the effectiveness of software diversity for coping with these attacks.

Acknowledgements

This research is supported by National Key Research and Development Program of China (2017YFB0803202).
References

[1] W. Jiangxing, Introduction to CCyberspaCe Mimic Defense. Science Press, 2017.
[2] “China national vulnerability database.” http://www.cnvd.org.cn/webinfo/list?type=5.
[3] A. J. O’Donnell and H. Sethu, “On achieving software diversity for improved network security using distributed coloring algorithms,” in Proceedings of the 11th ACM conference on Computer and communications security, pp. 121-131, ACM, 2004.
[4] C. Huang, S. Zhu, Q. Guan, and Y. He, “A software assignment algorithm for minimizing worm damage in networked systems,” Journal of Information Security and Applications, vol. 35, pp. 55-67, 2017 (in Chinese).
[5] Z. Zheng, M. Bolin, and W. Jiangxing, “The test and analysis of prototype of mimic defense in web servers,” Journal of Cyber Security, vol. 2, pp. 13-28, 2017 (in Chinese).
[6] S. Volckaert, B. Coppens, and B. De Sutter, “Cloning your gadgets: Complete rop attack immunity with multi-variant execution,” IEEE Transactions on Dependable and Secure Computing, vol. 13, no. 4, pp. 437-450, 2016.
[7] F. Majorczyk and J.-C. Demay, “Automated instruction-set randomization for web applications in diversified redundant systems,” in Availability, Reliability and Security, 2009. ARES’09. International Conference on, pp. 978-983, IEEE, 2009.
[8] L. Xingguo, T. Qing, Z. Zheng, and W. Jiangxing, “Mimic defense technology,” Engineering Sciences, vol. 18, pp. 69-73, 2016 (in Chinese).
[9] S. Allier, O. Barais, B. Baudry, J. Bourcier, E. Daubert, F. Fleurey, M. Monperrus, H. Song, and M. Tricoire, “Multitier diversification in web-based software applications,” IEEE Software, vol. 32, no. 1, pp. 83-90, 2015.
[10] P. Larsen, A. Homsesc, S. Brunthaler, and M. Franz, “Sok: Automated software diversity,” in 2014 IEEE Symposium on Security and Privacy (SP), pp. 276-291, IEEE, 2014.
[11] B. Baudry and M. Monperrus, “The multiple facets of software diversity: Recent developments in year 2000 and beyond,” ACM Computing Surveys (CSUR), vol. 48, no. 1, p. 16, 2015.
[12] M. A. Hiltunen, R. D. Schlichting, C. A. Ugarte, and G. T. Wong, “Survivability through customization and adaptability: The cactus approach,” in discex, p. 0294, IEEE, 2000.
[13] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, “Analysis of operating system diversity for intrusion tolerance,” Software: Practice and Experience, vol. 44, no. 6, pp. 735-770, 2014.
[14] A. Gorbenko, V. Kharchenko, O. Tarasyuk, and A. Romanovsky, “Using diversity in cloud-based deployment environment to avoid intrusions,” in International Workshop on Software Engineering for Resilient Systems, pp. 145-155, Springer, 2011.
[15] J. Wu, “Mimic security defense in cyber space,” Secrecy Science and Technology, vol. 10, no. 1, pp. 4-9, 2014 (in Chinese).
[16] “Graph coloring.” https://en.wikipedia.org/wiki/Graph_coloring.
[17] D. J. Welsh and M. B. Powell, “An upper bound for the chromatic number of a graph and its application to timetabling problems,” The Computer Journal, vol. 10, no. 1, pp. 85-86, 1967.
[18] R. Mengmeng, Y. Sixing, and Q. Zhaowei, “Power and spectrum allocation in d2d networks based on coloring and chaos genetic algorithm,” Procedia Computer Science, vol. 107, pp. 183-189, 2017.
[19] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, “The internet topology zoo,” Selected Areas in Communications, IEEE Journal on, vol. 29, pp. 1765-1775, October 2011.
[20] “Networkx.” https://networkx.github.io/.