Journal of Information Security and Applications 35 (2017) 55–67

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

A software assignment algorithm for minimizing worm damage in networked systems

Chu Huang[a], Sencun Zhu[b], Quanlong Guan[c,*], Yongzhong He[d]

[a] College of Information Science and Technology, The Penn State University, USA
[b] Department of Computer Science and Engineering, The Penn State University, USA
[c] Network and Education technology center, Jinan University, China
[d] School of Computer and Information Technology, Beijing Jiaotong University, China

ARTICLE INFO

Article history:
Available online 30 May 2017

Keywords:
Software diversity
Heterogeneity
Software assignment
Graph coloring

ABSTRACT

Homogeneous networked systems are at high risk of being compromised by malicious attacks that exploit a single weakness common to all. Following the *survivability through heterogeneity* philosophy, we present a novel approach to improving survivability of networked systems via software diversity. In this work, we propose an algorithm for assigning a number of software packages over a network of systems in an intelligent way such that machines running identical software are isolated into small "islands", hence restricting the worm-like attacks from propagation. While developing the algorithm, we take into consideration not only practical constraints, including host functionality and software availability, but also weight, severity and impact range of vulnerability, well balancing and effectively minimizing the potential damage by an single attack. We also introduce possible enhancements by taking advantage of topological features of the network. Finally, we present a comparative analysis of our algorithm using simulation over various network structures. The results not only confirm the effectiveness and scalability of our algorithm, but also show its capability in creating moving attack surface. The level of heterogeneity our algorithm can actually create depends on the ratio of the number of installed software to the total number of available software.

1. Introduction

With the fast advancement of nowadays information technology, organizations are becoming ever more dependent on interconnected systems for carrying out everyday tasks. However, the pervasive interdependence of such infrastructure increases the risk of being attacked and thus poses numerous challenges to system security. One major problem for such networked environments is software monoculture [36,51] – running on the risk of exposing a weakness that is common to all of its components, it facilitates the spread of attacks and enables large-scale compromises. Considering the consequences of software monoculture in intensively connected systems, there is an urgent need to control the damage of such security compromises that take advantage of the connectivity of the networked system.

In contrast to homogeneous systems by software monoculture, heterogeneous architectures are expected to have higher survivability [45,59,62]. This point is very much like the maintenance of genetic and ecosystem diversity in biology. The variability in the biological world allows at least a portion of species to survive an epidemic. Inspired by such phenomena of biodiversity, a good number of techniques have been proposed to improve system resilience and survivability under attacks. However, previous approaches cannot fully meet two highly desired requirements: (R1) *Resistance* against widespread security compromises in a networked environment; (R2) *Practicability* of the solutions under real-world constraints. To see how existing approaches are limited in meeting these two requirements, we classify them into three categories: software diversity at the system level, software diversity at network level and N-version programming. 1) Diversity at the system level is achieved mainly through randomization techniques, which are limited to individual machines and it is not clear if and how they can be extended to improve the survivability of the networked systems as a whole. 2) Diversification methods at the network level compensate the limitations of the former approach, but they suffer from the problem of only considering single version assignment of software. In the real world scenarios, however, a host (i.e., a commodity PC) typically is required to install with more than one software (i.e., operating system, web browser,

* Corresponding author.
E-mail address: gql@jnu.edu.cn (Q. Guan).

http://dx.doi.org/10.1016/j.jisa.2017.05.004
2214-2126/

email client, office suite applications etc.) to perform particular tasks. 3) N-version programming: achieves higher system survivability depending on its underlying multiple-version software units that tolerate software faults. This method has very high computational cost and is not practical enough to be used routinely in real-world organizations.

In order to address the research gap, in our previous work [28] we proposed a method that utilizes diverse off-the-shelf software to create a heterogeneous environment for networked systems against worm-like attacks. In practice, common vulnerabilities are generally a result of common code or shared specifications. Based on the fact that most of the off-the-shelf software are developed independently by different groups of developers, they are unlikely to exhibit the same vulnerabilities [25]. Hence, by assigning different software packages to different systems in a network, it will increase the difficulty for an attacker to construct a spreading malware that exploits common vulnerabilities in the software.

In this work, we present a new graph multi-coloring algorithm for assigning software packages to hosts in order to isolate worm-like attacks within a small "island. Different from [28], we introduce the concept of severity of software and take into account a new dimension when performing the software assignment task. In reality vulnerabilities do not have equal impacts: A more severe vulnerability (e.g. gain root privilege) could result in a far more serious damage compared to a mild vulnerability (e.g. collect some types of user information). Assessing damage of the vulnerabilities and distributing it accordingly is beneficial for avoiding disproportionately large damage in the network. Therefore, in this work, we want to take a step further and propose an algorithm that is able to minimize the potential damaging impacts of the vulnerable clusters. Such impacts can more accurately reflect the ability of the attacker to compromise the networked system, rather than just the sizes of the clusters. Besides, we explore topological features of the network structure and show that by leveraging network topological features, our algorithm can further reduce the potential damage of the network resulted from a single attack.

Specifically, we make the following contributions.

- First, we model the software assignment problem as a graph multi-coloring problem with practical system requirements and constraints, and take into account of different severity impacts of vulnerabilities. We also design an efficient heuristic algorithm to compute a near optimal solution for our problem.
- Second, based on the proposed algorithm, we introduce two possible enhancements by taking advantage of topological features of the network structure. Interesting question answered include: *How will the critical nodes of a network be identified and influence the assignment solution?*
- Third, to understand what kinds of network structures facilitate diversity, we evaluate the algorithm on different graph topologies. Through experiments, we find that the level of heterogeneity in our algorithm depends on the ratio of the number of software installed to the total number of available software, and we also identify critical ratio points for different representative topologies. Our findings may give some practical guidelines for choosing appropriate system parameters for balancing the trade-off between survivability and the assignment cost.

The rest of the paper is organized as follows. Section 2 reviews major studies on software diversity in different dimensions and other related works on graph coloring. Section 3 illustrates how to model the software assigning task as a graph coloring problem. Section 4 introduce the software assigning algorithm in detail and propose two improving schemes that utilize the topological features of the network. Section 5 evaluates the performance of our software assigning strategies. Section 6 gives extensive discussions of related issues. Finally, Section 7 concludes our work and provides future research directions.

2. Related work

The state-of-the-art approaches on software diversity are classified into three main categories: diversity at the system level and the network level, and N-version programming.

Software diversity at the system level has been focused on address space layout randomization, instruction set randomization and data randomization. *Address space layout randomization* (ASLR), as a very successful technique [35,52], randomizes the base address of each program region: heap, code and stack. It has already been implemented in major operating systems [56], including OpenBSD, Linux, Windows, MacOS, Android and iOS. In [8], a leakage-resilient layout randomization for mobile and embedded devices was prposed. Another randomization technique for software transformation is the *instruction set randomization*(ISR). Portokalidis and Keromytis [34] proposed to obfuscate underlying system's instructions in order to defeat code-injection attacks. Their proposed method randomizes all binaries with different secret keys so that malicious code introduced by the attackers would fail to execute correctly. *Data randomization* is another randomized-based approach. By applying different random masks on data in the memory [11], it disrupts the attempt to write outside objects on the memory since attackers cannot determine memory regions that are associated with particular objects. Cowan et al. [15] presented an approach that randomizes stored pointer values, as opposed to the locations where objects are stored. The encryption is achieved by XORing pointer values with a random integer mask. These works rely on attackers' inability to guess a secret key for security. It is not clear whether software transformation of individual machines would lead to overall diversification of the networked system as a whole.

The idea of just-in-time compilation randomization has been studied by Homescu et al. [27]. Their approach neither creates diverse versions of the same program nor introduces randomization points: the randomization happens in the just-in-time compiler directly. In [38], Liu et al. explored a hardware-level approach at micro instruction level and presented a diversification technique that remaps machine code of the same ISA into multiple variants based on programmable decoder. In [37], the authors systematically studied the state-of-the-art in system-level software diversity and highlighted fundamental trade-offs between fully automated approaches.

Diversity at the network level is achieved by using different applications, operating systems, and communication protocols [21,62] within a networked system. Mont et al. [40] introduced an approach to ensuring diversity for common, widespread software applications in which diversity is enforced at the installation time by a random selection and deployment of critical software components. Hiltunen et al. [26] proposed the use of fine-grained customization and dynamic adaptation as the key enabling technologies to achieve the goal of survivable system design. O'Donnell and Sethu [45] presented several distributed algorithms for assigning different versions of software to individual systems. However, their algorithms require high communication overhead when negotiating colors among the nodes. Yang et al. [59] highlighted the same diversity idea and applied it in the sensor networks with limited choices of software versions. The work was later extended in [60] by allowing each sensor node to be pre-loaded with more than one diversitified version of the same software. However, the effectiveness of these approaches on complex networks is still vague. Moveover, the works in [45,59,60] only address the problem of assigning multiple versions of a single software package. In reality, every machine is installed with many software packages and