PAPER

Designing and Implementing a Diversity Policy for Intrusion-Tolerant Systems

Seondong HEO†, ‡, Soojin LEE††, Nonmembers, Bumsoon JANG‡, Student Member, and Hyunsoo YOON†, Nonmember

SUMMARY  Research on intrusion-tolerant systems (ITSs) is being conducted to protect critical systems which provide useful information services. To provide services reliably, these critical systems must not have even a single point of failure (SPOF). Therefore, most ITSs employ redundant components to eliminate the SPOF problem and improve system reliability. However, systems that include identical components have common vulnerabilities that can be exploited to attack the servers. Attackers prefer to exploit these common vulnerabilities rather than general vulnerabilities because the former might provide an opportunity to compromise several servers. In this study, we analyze software vulnerability data from the National Vulnerability Database (NVD). Based on the analysis results, we present a scheme that finds software combinations that minimize the risk of common vulnerabilities. We implement this scheme with CSIM20, and simulation results prove that the proposed scheme is appropriate for a recovery-based intrusion tolerant architecture.

key words: intrusion-tolerant system, diversity, virtual machine, vulnerability, software

1. Introduction

Improvements in networking technologies and the rapid dissemination of smartphones new provide easy access to useful information services for a variety of purposes, such as performing financial transactions, social networking, and searching via the Internet. The enhanced accessibility provides convenience and efficiency to end users, but it also provides opportunities to attackers to intrude and compromise information systems. Although traditional protection techniques, such as intrusion detection systems (IDSs) and firewalls are used to protect information systems from external attacks, those approaches cannot detect and prevent all kinds of attacks, and information systems can still be successfully attacked. Critical information systems must provide reliable services in spite of successful attacks. Therefore, the concept of intrusion tolerance is more suitable for the protection of critical systems than traditional protection approaches.

There have been many ITS studies to enhance the reliability and survivability of ITSs, and most of those studies have been based on redundancy and diversity. Since critical systems must provide a proper service even if some components of systems are compromised, redundancy is a mandatory mechanism to eliminate the SPOF problem and increase the safety of critical systems. Additionally, redundancy is applied to majority-voting systems which produce a single output from a number of unreliable results. However, if a system employs the same software or hardware to implement redundant functions, the components of the system will have common vulnerabilities and weaknesses. Diversity can address this problem through the implementation of components of the system in a variety of ways.

Studies on Byzantine fault-tolerance have applied the diversity principle to guarantee correct operation of a critical system despite failures of a minority of components in the system [1]. However, implementing diversity become more complicated when security is considered. If the components of a system share the same vulnerabilities, they must have the same weaknesses to the same attacks. Attackers prefer to exploit these vulnerabilities because attacks that exploit these vulnerabilities can compromise multiple components simultaneously.

Most current information systems which provide essential services such as banking, social network, and search use commercial off-the-shelf (COTS) software for many reasons, such as rapid availability, cost reduction, and easy maintenance [2]. Unfortunately, security is not the highest priority for COTS vendors. In general, COTS software are designed to meet functional requirements rather than security requirement. COTS vendors reuse templates, functions, and procedures to save time and cost for developing software. This means that the software produced in this way share security flaws that can be exploited. In particular, there are a lot of common weaknesses within families of software (e.g., Windows, Unix). Nevertheless, people use COTS software rather than making their own because the current software used to build information systems, such as operating system (OS) and database management system (DBMS), is too complex for individuals to develop by themselves. Additionally, there would be no guarantee that their developed software would be more secure than COTS software.

For these reasons, current information systems are built with COTS software. When systems adopt COTS software, there are two approaches to reduce common vulnerabilities: developing variants of software and using diverse software. The latter is less complex and expensive than the former.

Copyright © 2017 The Institute of Electronics, Information and Communication Engineers
Also, since variants of software must have the same code for compatibility, they still share some vulnerabilities in spite of modification. Therefore, we use diverse software and virtualization technology to achieve the goal of intrusion tolerance.

Virtual machines (VMs) are very attractive for information systems. They provide high resource utilization and fault isolation that can consolidate the security of systems. Additionally, virtualization technology allows multiple servers to operate with different configurations on a single physical machine. Using VMs is not only effective to implement diversity, but also to secure systems. A proactive recovery technique which resumes VMs in a pristine state periodically can eliminate the effects of malicious attacks [3], [4]. If the cleansing procedure is performed often enough, then attackers do not have enough time to compromise a sufficient number of VMs to break down the whole system. The cleansing procedure loads a clean image containing the OS and applications into VMs.

The proactive recovery technique reduces the damage caused by attacks, even if they are stealth attacks that are undetectable. However, although the system employs proactive recovery, attackers can simultaneously compromise many or all of the VMs in a critical system by exploiting common vulnerabilities. This problem must be resolved because intrusion tolerance is generally applied to critical systems, which have to survive in the face of intrusions. ITSs based on proactive recovery have mitigated the problem by using diverse software.

In this paper, we address the issue of how to select the best configuration among a pool of COTS software. To answer the question, we have collected vulnerability data from the NVD reported between 2008 and 2014 [5]. Among various vulnerabilities in the NVD, we analyzed the vulnerabilities related to Web services, especially OSs, Web server software, and database management systems. There are several reasons why we focused on these types of software. First, Web technologies have been applied to a wide range of fields from commercial to military systems. Therefore, protecting Web servers has been one of the most interesting topics in ITS studies [6], [7]. Second, the types of software that are used to provide Web services provide an opportunity for diversity. Additionally, there are a number of vulnerabilities related to software; therefore, we can obtain meaningful results from analyzing the vulnerabilities. In particular, we concentrated on vulnerabilities that affect more than one software.

We also introduce a scheme that finds the best combinations that minimize the risk of common vulnerabilities. The scheme recommends a configuration for a VM being recovered. The scheme considers not only common vulnerabilities but also software exposure patterns to make critical information systems more secure. We implement the scheme by using CSIM20 to evaluate the effectiveness of our suggested scheme [8]. Our experiments indicate that the proposed scheme is effective in enhancing the availability of the system, and it may affect the confidentiality and the integrity of the system.

The rest of this paper is organized as follows. Section 2 summarizes the related work on an ITS. In Sect. 3, we analyze the vulnerabilities that affect operating systems, database management systems, and Web server software. Section 4 describes the proposed schemes, and Sect. 5 presents the experimental design and results. The paper ends with a conclusion in Sect. 6.

2. Related Work

Design diversity was introduced in the 1970s for fault tolerance [9]. Randell suggested using redundant components which were implemented independently and performed the same function. Chen and Avizienis presented N-version programming which generates functionally equivalent programs [10]. N-version programming uses N different implementations of the components, which are independently developed by N groups who use different algorithms and languages. Although N-version programming is helpful to achieve fault tolerance, N-version programming is too costly to use in practical systems. The use of diverse off-the-shelf (OTS) products is a cost effective means to improve system reliability [11]. The authors analyzed bug reports for four popular OTS DBMS products and checked for bugs that could affect more than one server. There were no coincident failures in more than two servers, and their experimental results suggest that diverse redundancy can improve the dependability of systems.

M. Garcia et al. carried out a study with OSs vulnerability data from the NVD [12]. The authors analyzed the vulnerabilities of 11 OSs to find the number of vulnerabilities that occur in more than one OS. The results suggest that using diverse OTS OSs is an effective method to secure a whole system.

Studies on ITS have adopted diverse COTS components because this practice is an effective way to increase security. Hierarchical adaptive control of quality of service for intrusion tolerance (HACQIT) architecture includes diverse replication and network-based IDS [13]. The author assembled the architecture with COTS software to provide service during several hours of cyber attacks. HACQIT uses two different software components, namely, Microsoft’s Internet Information Server (IIS) and the Apache Web server, to detect errors and mask failures.

Scalable intrusion-tolerant architecture (SITAR) consists of five main components to defend COTS servers from attacks [14]: proxy server, ballot monitor, acceptance monitor, adaptive reconfiguration module, and audit control module. An adaptive reconfiguration module adaptively reconfigures the overall system when the audit control module detects intrusions to provide a minimal level of services.

The architecture of dependable intrusion tolerance (DIT) consists of redundant HTTP COTS servers which run diverse operating systems and platforms [15]. DIT employs proxies and a monitoring system to mediate client request and verify the behavior or server and other proxies.
Designing protections and adaptation into a survivability architecture (DPASA) has various layers to include approaches of intruders and improve system reliability [16]. The architecture integrates elements of protection, detection, and adaptive reaction. Adaptive response adds runtime diversity by changing the system configuration or behavior. Also, employing other cyber mechanisms is possible to improve the survivability of the architecture [17].

A generic architecture for intrusion-tolerant Web servers is composed of redundant Web servers and proxies [7]. The authors used diversification to increase resilience to attacks and independence between redundant components. The diversified Web servers provide the same services, but run different application software and O/Ss on diverse hardware platforms. Tolerance proxies that are included in the architecture have essential functions: mediating client requests, monitoring the Web servers and the other proxies, protecting the database, and selecting the regime. Diverse O/Ss and platforms are used for these proxies. The authors employed IDS which utilizes diversified complementary detection mechanisms.

The common problem of the above studies is that they depended on intrusion detection techniques. If malicious attackers have enough time, they discover and exploit vulnerabilities that cause the collapse of the whole system. Proactive recovery allows the removal of intrusions from compromised replicas without intrusion detection. Self-cleansing intrusion tolerance (SCIT) is a representative technique that utilizes proactive recovery [18]. The SCIT architecture consists of a central controller and redundant virtual servers. The state transitions of each VM are performed by the central controller. The states are the following:

(i) Active: Server receives and processes requests from the external network.
(ii) Grace period: Server processes accepted requests, but does not receive any more requests.
(iii) Cleansing period: Server is isolated from the external network, and recovered to a pristine state.
(iv) Live spare: Server is in a pristine state, and waits to be active.

P. Sousa et al. proposed an architecture which uses not only proactive recovery, but also reactive recovery [4]. Correct replicas in the architecture can force the recovery of a replica that is detected or suspected of being faulty. Each replica consists of two parts: payload and wormhole. The wormhole performs communication between VMs’ wormholes and reactive recovery. The architecture employs diversity to avoid common failures, such as different O/Ss with different configurations (e.g., root passwords, kernel versions).

3. Analyzing Vulnerabilities

This section presents how we selected and analyzed our data set. We collected and analyzed vulnerability data from the NVD reported between 2008 and 2014 [5]. The NVD uses the common vulnerabilities and exposures (CVE) definition of vulnerability [19]; an information security “vulnerability” is a mistake in software that can be directly used by a hacker to gain access to a system or network. The CVE guidelines consider a mistake to be a vulnerability if it allows an attacker to use it to violate a reasonable security policy for that system, excluding “open” security policies in which all users are trusted, or where there is no consideration of risk to the system. In CVE, a vulnerability is a state in a computing system that does one of the following:

- allows an attacker to execute commands as another user
- allows an attacker to access data that is contrary to the specified access restrictions for that data
- allows an attacker to pose as another entity
- allows an attacker to conduct a denial of service

The NVD contains publicly known information about security vulnerabilities and exposures. Data feeds that contain all the reported vulnerabilities are provided as XML files. Each vulnerability in the NVD has a unique identifier that enables data exchange between security products and provides a baseline index point for evaluating the coverage of tools and services. Each CVE identifier includes the following: CVE identifier number, brief description of the security vulnerability or exposure, and any pertinent references. The syntax of CVE identifier (CVE-ID) is variable in length and includes the following: CVE prefix-year-arbitrary digits (e.g., CVE-2014-0001). Arbitrary digits can be expanded from 4 digits when needed. In addition to the above data, CVE entries have other useful information, such as the list of products that are affected by the vulnerability, the date of the vulnerability publication, the common vulnerability scoring system (CVSS) score, and security metrics.

Though there is a large amount of vulnerability data in the NVD, we were only interested in OS, DBMS, and Web server software. The common platform enumeration (CPE) describes and identifies classes of applications, operating systems, and hardware devices among an enterprise’s computing assets [20]. We selected products according to the CPE specification and market share as follows [21]–[23]:

- Operating system: Windows Server 2003, Windows Server 2008, Debian, Ubuntu, FreeBSD, and NetBSD
- Web server software: Apache, Nginx, Microsoft-IIS, and LiteSpeed
- Database management system: Oracle, MySQL, Microsoft SQL Server, and MongoDB

In addition, each CVE entry provides meaningful information, but we focused on the CVE-ID, publication date, CVSS score, and the list of affected products. After collecting the vulnerability data from NVD, we developed a program in Python that parses the XML data feeds and selects entries related to products and information in which we were interested. Our parser made two tables from the NVD data to manipulate the vulnerability data efficiently.

CVSS scores are used to reflect the impacts of vulner-
abilities [24]. CVSS consists of three groups: Base, Temporal and Environmental. Each group produces a numeric score ranging from 0 to 10, and a Vector, a compressed textual representation that reflects the values used to derive the score. Basically, CVSS considers the intrinsic and fundamental characteristics of a vulnerability that are constant over time and user environment. Additional information can be added to reflect the risk according to the environment. A high CVSS score means that exploiting the corresponding vulnerability requires low complexity, and it causes serious damage to information security (i.e., confidentiality, integrity, availability).

3.1 Operating System Vulnerabilities

Table 1 shows the distribution of vulnerabilities of the analyzed OSs, with the total CVSS scores of vulnerabilities. An important observation in Table 1 is that the sum of the columns is not same as the last row of the table. This means that some vulnerabilities affect more than one OS. Thus, we were able to find the number of common vulnerabilities and CVSS scores to identify OSs that share a minimum weaknesses.

Table 2 shows the common vulnerabilities and CVSS scores that were found in the pool of OSs between 2008 and 2014. The numbers in parentheses are the sums of CVSS scores of the corresponding common vulnerabilities. The diagonal cells of the table show the vulnerability information of the OS.

We obtained two meaningful results from Table 2. First, the number of common vulnerabilities between two OSs is smaller than the number of vulnerabilities of an OS. That is, a system built with multiple OSs is more resilient to attacks than a system built with a single OS. Therefore, attackers who want to intrude upon the former system have to exploit many vulnerabilities to compromise the whole system. On the other hand, the latter system can be broken down by exploiting a small set of vulnerabilities.

Second, a system that deploys OSs belonging to different OS families is more secure than a system which uses a single OS or multiple OSs from the same family. Over the observed period, the Windows server family had the highest number of shared vulnerabilities. These vulnerabilities were caused by the software components and the applications, which were reused in the family. On the other hand, the Linux kernel had the highest number of vulnerabilities (819) over the observed period, although the number of common vulnerabilities between Linux distributions was much smaller than that of the Windows family. Therefore, we suspect that developers of Linux distributions have modified their kernels, software components, and applications.

3.2 Web Server Software Vulnerabilities

Table 3 shows the common vulnerabilities and CVSS scores that were observed in the data from 2008 and 2014. The total number of Web server software vulnerabilities was much smaller than the number of OS vulnerabilities. Since an OS is the most important and most complex software, large codes are needed to implement the functions of an OS, and large codes generally have several programming flaws.

The number of shared vulnerabilities was also smaller than the number of common vulnerabilities between OSs. Because each Web server software was developed by different groups, they shared a small number of weaknesses.
3.3 Database Management System Vulnerabilities

Table 4 shows the common vulnerabilities and CVSS scores that were found in the pool of OSs between 2008 and 2014. A noticeable observation is that the total number of DBMS vulnerabilities was much larger than the number of Web server software vulnerabilities. The discovery of vulnerabilities can be based on coincidental detection or active search by persons who want to gain fame or economic profit. Since databases contain critical and sensitive information, they are very attractive targets of individuals with intrinsic or extrinsic motivations. Another interesting observation is that there is no common vulnerability between DBMS in contrast to OS and Web server software. Therefore, using diverse DBMS can secure a system without anxiety about common vulnerabilities.

3.4 Common Vulnerabilities of Different Types of Software

Though vulnerabilities can affect a variety of software, most of the vulnerabilities mentioned in 3.1, 3.2, and 3.3 affect only OS, Web server software, or DBMS, respectively. However, some vulnerabilities can influence products different types of products. For example, CVE-2008-3013, which allows attackers to execute arbitrary code via a GIF image file, affect not only Windows Server 2008, but also Microsoft SQL Server and other products (e.g., Microsoft Office, Visio). Table 5 shows the common vulnerabilities of different types of software.

4. Selecting Configurations

An ITS must continue provide proper services under a threatening environment. To achieve this goal, recent studies on ITSs have utilized virtualization technology to build an intrusion-tolerant replicated system. Virtualization technology allows proactive recovery to remove undetected effects of attacks. Additionally, replicated systems can be built with diverse software by using VMs. However, as discussed in Sect. 3, it is not appropriate for ITS to just use a number of different software.

In this paper, we introduce a configuration selection algorithm. Each configuration includes an operating system, a database management system, and Web server software for VMs. We select a configuration that satisfies the following two conditions:

- Minimizing the sum of CVSS scores of shared vulnerabilities
- Randomizing exposure patterns of configurations

Traditional proactive recovery techniques recover VMs by using pristine images, which are saved in a protected storage. Most recovery-based ITSs adopt a proactive recovery technique to remove malicious intrusions. However, if each VM in a system selects a configuration without consideration of common vulnerabilities, the system can be severely damaged by attackers who exploit these vulnerabilities. For example, there are 375 common vulnerabilities between Windows Server 2003 and Windows Server 2008. If an administrator builds a system with these products, attackers can compromise the system by exploiting common vulnerabilities, which cause damage to a number of replicas. Therefore, selecting types of software which have the minimum number of common vulnerabilities is necessary to build a secure and reliable system.

Ideally, the VMs in a system run different configurations, and there is no shared vulnerabilities between active VMs. Also, if the number of active VMs is small and the system has an enough configurations, finding such configurations is entirely possible. Unfortunately, we have a limited number of configurations, and critical systems typically run a number of VMs simultaneously to provide stable services. Therefore, the selection algorithm should be able to select a new configuration, which share the least number of vulnerabilities with configurations of other active VMs, according to system environments.

The number of common vulnerabilities can be used to evaluate the security of a system, but each vulnerability has unique characteristics in terms of exploitability, attack complexity, and impact. Each vulnerability entry in the NVD has a CVSS score, and the CVSS score reflect the properties and dangers of the vulnerability. Therefore, we use CVSS scores of common vulnerabilities to measure the security of a system.

To find the the best configurations that minimize the risk of common vulnerabilities, we generate all the possible configurations and compare the sum of the CVSS scores of vulnerabilities which are shared by configurations. When we want to find the best configurations which consists of $k$ configurations, this problem can be formulated as follows.

**Problem 1.** Instance: A number $k$, a collection of configurations $C = C_1, C_2, \ldots, C_m$, and a collections of vulnerabilities...
$V = V_1, V_2, \ldots, V_m$. $V_i$ means the collection of the vulnerabilities that occur in $C_i$.

Objective: Find a subset $C' \subseteq C$ of configurations, such that $|C'| = k$ and the number of shared vulnerabilities $\left| \bigcup_{C_i \in C'} (V_i \cap V_j) \right|$ is minimized.

$\sum_{C_i \in C'} (V_i \cap V_j) = \left| \bigcap_{C_i \in C'} (V_i \cap V_j) \right|$, where $c$ is the complement operation. Therefore, finding a subset that minimizes the expression is the minimum intersection problem which is polynomial time equivalent to the maximum coverage problem [25]. The maximum coverage problem is NP-hard, and the greedy algorithm is the best-possible polynomial time approximation algorithm [26]. Therefore, we use the greedy algorithm to implement Algorithm 1.

Algorithm 1 works without initial configurations, we find the initial configurations that minimize the sum of the CVSS scores of the common vulnerabilities to improve the performance of Algorithm 1 before the initialization phase. We use integer linear programming to find the configurations [27].

Although a system uses diverse configurations that share the minimum number of vulnerabilities, if a proactive recovery is performed without consideration of exposure patterns, the following several problems can occur:

- As VMs are repeatedly exposed to the external network, the vulnerabilities of each VM are exposed repeatedly, too. Attackers can easily detect and exploit these vulnerabilities to intrude upon each VM.
- Pristine images have their own vulnerabilities Therefore, if attackers detect the exposure patterns, they will discover vulnerabilities that are shared by VMs in the online state. In this case, attackers can break down the whole system by exploiting these common vulnerabilities.

To mitigate these problems, we generate and use a random number to hide the rotation pattern and configurations of VMs that are providing services. When a VM expires the exposure timeout, the central controller assigns a random number to the configuration of the VM. When the central controller selects a configuration for a VM, the central controller selects the configuration that has a low assigned number and decrease all assigned numbers of configurations by 1. The BackOff variable in algorithm 1 is used to represent the random number.

Algorithm 1 is run by the central controller, and it selects a configuration for a VM which will be exposed to the external network. Algorithm 1 consists of three functions: TerminateVM(VM), SelectConfiguration(), and GetSharedCVSS(CONF). Function TerminateVM(VM) terminates a VM which expires the exposure timeout and assigns a BackOff value to the configuration of the VM. The SelectConfiguration() and GetSharedCVSS(CONF) functions are used to select the most appropriate configuration. The configuration is picked from the configuration pool (CONFPool), and it satisfies the above conditions. In the initialization phase (Lines 1-7), the algorithm initializes some global variables. The system administrator assigns an appropriate value to $B$ according to the size of the configuration pool. The OSPool, WSSPool, and DBMSPool contain the information of the OS, Web server software, and DBMS, respectively. Here, $l$, $m$, and $n$ indicate the size of each software group. CONFPool is the set of configurations that can be built with the OSPool, WSSPool and DBMSPool. The OPool contains the information of configurations used by active VMs.

When the central controller terminates a VM that is exposed to the external network for a certain period of time, the algorithm generates a random number (Line 8) and ap-
plies mod-operation to suspend exposure of the configuration of the terminated VM. CONFid presents the identification number of the configuration. After removing the configuration of the VM from the OPool, the algorithm assigns the generated random value to the corresponding configuration in the CONFPool.

The SelectConfiguration() function determines which configuration should be used. The function makes configurations which can be built with COTS products that belong to the system (Lines 15-21). Each configuration is a candidate for NCONF, which will be recommended for a VM that will provide a service. NCONF must fulfill two conditions; the sum of the CVSS scores of vulnerabilities which are shared by configurations (CVul) in OPool must smaller than any other configuration, and Backoff should be 0. Here, CVul can be obtained from the GetShared(CONF) function.

The GetShared(CONF) function calculates retCVSS, which indicates the sum of the CVSS scores of shared vulnerabilities between CONF and configurations run by VMs in the processing group. Although Table 7 contains data that have been collected between 2008 and 2014, new vulnerabilities can be considered with little effort by using our parser. This feature makes ITSs more secure against threats which are being exposed every day.

### 5. Experimental Design and Results

In this section, we present the architecture of the ITS which employs the configuration selection algorithm. Our prototype was designed for Web servers, which provide services and contents that can be accessed via the Internet. The proposed system is depicted in Fig. 1. All VMs belong to a cleansing group or a processing group. VMs in the cleans-

---

| Product ID | The number of uses of the product |
|------------|----------------------------------|
|            |                                  |
| 1          | 5                                |
| 2          | 4                                |
| 3          | 5                                |
| ...        | ...                             |
| p + q + r  | 4                                |

| ID   | 1       | 2       | 3       | 4       | 5       | 6       | 7       | 8       | 9       | 10      | 11      | 12      | 13      | 14      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 1    | 3259.8 | 2851.5 | 0      | 0      | 14.3   | 14.3   | 0      | 0      | 9.0    | 0      | 0      | 0      | 0      | 74.4    |
| 2    | 2851.5 | 3663.5 | 0      | 0      | 14.3   | 14.3   | 0      | 0      | 9.0    | 0      | 0      | 0      | 0      | 97.7    |
| 3    | 0      | 0      | 601.3  | 81.9   | 0      | 0      | 0      | 0      | 0      | 10.2   | 0      | 0      | 0      | 0       |
| 4    | 0      | 0      | 81.9   | 977.5  | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 4.3     |
| 5    | 14.3   | 14.3   | 0      | 0      | 488.2  | 107.6  | 4.3    | 0      | 0      | 0      | 0      | 0      | 0      | 0       |
| 6    | 14.3   | 14.3   | 0      | 0      | 107.6  | 181.9  | 4.3    | 0      | 0      | 0      | 0      | 0      | 0      | 0       |
| 7    | 0      | 0      | 0      | 0      | 4.3    | 4.3    | 289.6  | 0      | 12.6   | 0      | 0      | 0      | 0      | 0       |
| 8    | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 102.0  | 0      | 0      | 0      | 0      | 0      | 0       |
| 9    | 9.0    | 9.0    | 0      | 0      | 0      | 0      | 12.6   | 0      | 175.3  | 0      | 0      | 0      | 0      | 0       |
| 10   | 0      | 0      | 10.2   | 0      | 0      | 0      | 0      | 0      | 0      | 85.0   | 0      | 0      | 0      | 0       |
| 11   | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 1547.3 | 0      | 0      | 0       |
| 12   | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 1167.6 | 0      | 0      | 0      | 0       |
| 13   | 0      | 0      | 0      | 4.3    | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 73.3   | 0       |
| 14   | 74.4   | 97.7   | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 198.5   |
ing group do not receive requests and make its state pristine. VMs in the processing group accept and process requests from clients. The direction of the arrows indicates the direction of communication. VMs in the processing group cannot communicate with the central controller, because they are potentially compromised.

We adopted a proactive recovery mechanism to protect the system from undetected attacks. The central controller periodically rotates the state of each VM, active → cleansing → ready → active, sequentially. The central controller runs the selection algorithm between cleansing and ready state.

We applied the following assumptions in designing the architecture.

- Since the internal and external networks are separated, the central controller is safe from attacks from the external network.
- Active VMs can be attacked, and a successful attack exploits vulnerabilities and compromises a certain VM.
- Other malicious behaviors to interrupt the operation of the system are not considered to simplify the evaluation.

The CSIM 20 simulator is used to estimate the efficiency of the proposed scheme [8]. By using the CSIM 20 simulator, we simulated the proposed scheme and conventional schemes. The experimental environments were determined as follows:

- Probability functions that are supported in CSIM 20 were used to provide the various request generation and processing times.
- The request generation times and processing times followed an exponential distribution.
- Every VM exposed to the external network was vulnerable to attacks. The number of attacks followed a Poisson distribution with \( \lambda \).
- Attacks compromise a VM. The total damage of the system depended on CVSS scores and the remaining exposure time.
- Cleansing procedures eliminated the effects of attacks from the corrupted VM.

We set \( \lambda \) to 0.00253 to create a realistic environment [28]. According to survey results, not all discovered vulnerabilities have been exploited. In the experiments, each successful attack exploited 10% of the vulnerabilities that affected the configuration of an attacked VM to exploit all the other VMs [29]. All VMs in the processing group that shared vulnerabilities were affected by an attack. The exposure time and cleansing time of VMs in the system were set to 120 seconds, and the average processing time is set to 0.01 second [30]. All VMs in the processing group have different processing time which is determined by the software combinations used in the VMs [31]–[33]. These values were commonly used throughout the experiments.

We compared the proposed scheme with a random selection scheme and a fixed configuration scheme. The random selection scheme randomly selects COTS products to organize a configuration. The proposed scheme and the random selection scheme do not select a configuration which cannot be used due to compatibility issues. All VMs in a system which applied the fixed configuration scheme used the same configuration, which consisted of NetBSD, Nginx, and MongoDB in Sect. 5.2, and Debian, Apache, and Oracle in Sect. 5.3, respectively. The former configuration has the best processing performance, and the latter configuration is the most commonly used for Web servers.

5.1 COTS Product Selection

Table 8 shows the most selected COTS products in each software group. When the size of processing group was 5, the numbers of times that NetBSD, FreeBSD, Debian, and Ubuntu were selected were similar. Since the size of the processing group was smaller than the size of the OS pool, the above four OSs were always used for active VMs, and one VM used an OS belonging to the Windows family. Lighttpd and MongoDB were most commonly selected. They were used much more frequently than other COTS products because the numbers of Web server software and DBMS were smaller than 5. They showed the fewest vulnerabilities in each software group. The most used configuration was (Win2003, IIS, MSS).

When the size of the processing group was 7, NetBSD was used more than other OSs. Although the sum of CVSS scores of common vulnerabilities was slightly larger than OSs that belonged to Linux family, it had the fewest vulnerabilities. Lighttpd and Nginx were mostly used in Web server software. MongoDB was used more than MSS, but the difference was very small. The most used configuration was (NetBSD, Nginx, MongoDB).

When the size of the processing group was 9, since the
number of active VMs was certainly larger than the size of the OS pool, the four OSs were used in a similar number. As in the above case, two COTS products were most commonly selected from Web server software and DBMS. The most used configuration was (Win 2003, IIS, MSS). As shown in Table 2, Win 2003 is not the most secure OS. However, the proposed scheme uses OSs in the Windows family to improve the security of the whole system. Additionally, Win 2003 was selected with only IIS, Lighttpd, MSS, and MongoDB, whereas OSs in the Linux family or BSD family were used with different Web server software and DBMS. For these reasons, the configurations including Win 2003 were used most when the size of the processing group was 7 or 9, although they are not the most secure configurations.

As a result, COTS products that have small numbers of vulnerabilities and shared vulnerabilities were selected by the proposed scheme. However, the objective of this research was not only find secure configurations, but also to make ITS more resilient to threats by using diverse COTS products. Therefore, the proposed scheme selected a configuration that included Windows family OSs, even though they had the largest number of vulnerabilities, and the sum of CVSS scores of shared vulnerabilities was much larger than that of other products.

5.2 Throughput under a DoS Attack

In this experiment, we evaluated the throughputs of intrusion-tolerant systems while they were being attacked by an attacker. The attacker launched a denial of service (DoS) attack against the systems every 500 seconds. The attack exploited the vulnerabilities of the target VM, and other VMs that shared the vulnerabilities were affected by the attack, too. Compromised VMs were not able to process requests for the remaining exposure time.

Figures 2, 3, and 4 show the throughputs of the systems. An ideal line represents the throughput of an ideal system which is unaffected by DoS attacks. We measured the number of requests processed by the systems every 20 seconds to obtain the throughput of the systems, and showed the averages of the throughputs of the last 200 seconds. The proposed, random, and fixed lines show the throughputs of the systems which employed the proposed scheme, a random selection scheme, and a fixed configuration scheme, respectively.

The throughputs of the systems (except the ideal case) decreased during attacks, and the effects of the attacks were removed by proactive recovery. VMs in the system that employed the fixed configuration scheme used configurations that shared the same vulnerabilities, and the throughput was significantly decreased. The throughputs of the systems that used diverse configurations also decreased, but the impact of a DoS attack was relatively small because VMs in the systems had different vulnerabilities. Unfortunately, the systems did not have an enough configurations to build an ideal environment in which individual VMs would run different configurations that have no common vulnerabilities. Therefore, attacks compromised more than one VM in the systems, and the number of compromised VMs increased when
the size of the processing group increased.

The random selection scheme sometimes made the system resilient to attacks; however, it does not consider shared vulnerabilities between VMs; therefore it cannot guarantee that the number of uncompromised VMs which can properly process requests is sufficient to provide the appropriate level of service.

Tables 9 and 10 show the average throughputs and minimum throughputs of the systems, respectively. Each number in the first row indicates the size of the processing group. As shown in Table 9, the throughput of the system that applied our scheme was higher than that of the random selection scheme on average. The difference becomes more obvious when we compare the minimum throughputs. Each cell of Table 10 contains the average of minimum throughputs caused by DoS attacks. The throughput loss of Proposed was 49.7% which was much less than those of Random (71.1%) and Fixed (85.2%).

The average throughput of the system that employed the fixed configuration scheme was higher than that of the proposed scheme, because the fixed configuration had the best performance among the configurations. However, the objective of intrusion tolerance is to maintain an acceptable level of service. In this regard, the proposed scheme is more appropriate for ITSs than the other schemes, because the system that applied our scheme showed much better performance than the other schemes during a DoS attacks as shown in Table 10.

5.3 Expected Damage of Systems

Figure 5 and Table 11 show the expected damage caused by attacks. The expected damage of an attacked VM can be determined by the product of the sum of CVSS scores of exploited vulnerabilities and the remaining exposure time. The total damage is the sum of the expected damage of all attacked VMs.

The damage incurred by the system based on our scheme would smaller than that of other schemes. As the size of the processing group increases, the differences between the schemes increase. In particular, the damage incurred by the system based on the proposed scheme is only 24.3% of that of the system based on the fixed configuration scheme when the size of the processing group is 9. These results demonstrate that the proposed scheme reduces the risk of common vulnerabilities in threatening environments.

6. Conclusions

Most current information systems that provide useful services are connected to the Internet, and it is hard to protect such systems against all threats present on the Internet. In this context, studies on ITS employing proactive recovery have been conducted to improve the security of such systems. However, when replicas of a system use the same software, the replicas have common vulnerabilities that can be exploited to collapse the entire system.

In this paper, we analyzed the vulnerabilities found in NVD data of 14 types of COTS software, which are used widely to provide Web services. We also introduced a configuration selection scheme to reduce the risk of common vulnerabilities. The proposed scheme makes information systems more secure by randomization of software exposure patterns. Though additional cost is required to implement and manage diversity, our experiments indicate that using the diverse components selected by the proposed scheme is an effective way to improve the intrusion tolerance of systems.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MEST) (No.2014R1A2A2A01006957) and Institute for Information & communication Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.
References

[1] L. Lamport, R. Shostak, and M. Pease, “The byzantine generals problem,” ACM Transactions on Programming Languages and Systems (TOPLAS), vol.4, no.3, pp.382–401, 1982.

[2] J.S. Gansler and W. Luycshyn, “Commercial-off-the-shelf (cots): Doing it right,” tech. rep., DTIC Document, 2008.

[3] Q. Nguyen and A. Sood, “Realizing s-reliability for services via recovery-driven intrusion tolerance mechanism,” Dependable Systems and Networks Workshops (DSN-W), 2010 International Conference on, pp.176–181, IEEE, 2010.

[4] P. Sousa, A.N. Bessani, M. Correia, N.F. Neves, and P. Verissimo, “Highly available intrusion-tolerant services with proactive-reactive recovery,” Parallel and Distributed Systems, IEEE Transactions on, vol.21, no.4, pp.452–465, 2010.

[5] “National vulnerability database.” http://nvd.nist.gov/. Accessed: 2015-11-23.

[6] S.B.E. Raj and G. Varghese, “Analysis of intrusion-tolerant architectures for web servers,” Emerging Trends in Electrical and Computer Technology (ICTEETC), 2011 International Conference on, pp.998–1003, IEEE, 2011.

[7] A. Saidane, V. Nicomette, and Y. Deswarte, “The design of a generic intrusion-tolerant architecture for web servers,” Dependable and Secure Computing, IEEE Transactions on, vol.6, no.1, pp.45–58, 2009.

[8] H. Schwetman, “Csm19: a powerful tool for building system models,” Proceedings of the 33nd conference on Winter simulation, pp.250–255, IEEE Computer Society, 2001.

[9] B. Randell, “System structure for software fault tolerance,” vol.10, no.6, pp.437–449, 1975.

[10] L. Chen and A. Avizienis, “N-version programming: A fault-tolerance approach to reliability of software operation,” Digest of Papers FTCS-8: Eighth Annual International Conference on Fault Tolerant Computing, pp.3–9, 1978.

[11] I. Gashi, P. Popov, and L. Strigini, “Fault tolerance via diversity for off-the-shelf products: A study with sql database servers,” Dependable and Secure Computing, IEEE Transactions on, vol.4, no.4, pp.280–294, 2007.

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

[13] J. Reynolds, J. Just, E. Lawson, L. Clough, R. Maglich, and K. Levit, “The design and implementation of an intrusion tolerant system,” Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on, pp.285–290, IEEE, 2002.

[14] F. Wang et al., “SITAR: A scalable intrusion tolerance architecture for distributed server IEEE 2nd SMC information assurance workshop,” West Point, New York, 2001.

[15] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saidi, V. Stavridou, and T.E. Uribe, “An architecture for an adaptive intrusion-tolerant server,” Security Protocols, pp.158–178, Springer, 2004.

[16] J. Chong, P. Pal, M. Atigetchi, P. Rubel, and F. Webber, “Survivability architecture of a mission critical system: The dpasa example,” Computer Security Applications Conference, 21st Annual, p.10, IEEE, 2005.

[17] P. Pal, F. Webber, and R. Schantz, “The dpasa survivable jbi-a high-water mark in intrusion-tolerant systems,” Proc. 2007 Workshop Recent Advances in Intrusion Tolerant Systems, 2007.

[18] Y. Huang and A. Sood, “Self-cleansing systems for intrusion containment,” Proc. Workshop on Self-Healing, Adaptive, and Self-Managed Systems (SHAMAN), 2002.

[19] “Cve terminology.” https://cve.mitre.org/about/terminology.html. Accessed: 2015-11-23.

[20] “Common platform enumeration.” http://scap.nist.gov/
Soojin Lee received the B.S degree in Computer Science from Korea Military Academy in Seoul, Korea, in 1992, the M.S. degree in Computer Science from Yonsei University, Seoul, Korea, in 1996, and the Ph.D. degree in Computer Science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2006. He is currently a Professor of the Department of Computer Science and Engineering, Korea National Defense University (KNDU). His research interests include mobile network security, intrusion detection system, and cyber warfare.

Bumsoon Jang is a Ph.D candidate at School of Computing in Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. He received the B.S in Computer Science from Korea Military Academy in Seoul, Korea, in 2003 and the M.S. degree in Computer Science from Texas A&M University in 2007. His main researches include the system security, especially the design and analysis of recovery-based intrusion tolerant system using the virtualization technique.

Hyunsoo Yoon received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1979, the M.S. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 1981, and the Ph.D. degree in computer and information science from The Ohio State University, Columbus, OH, in 1988. He is currently a Professor at School of Computing in KAIST. His research interests include mobile ad hoc networks, wireless networks, and network security.