Garcia, M., Bessani, A. N., Gashi, I., Neves, N. & Obelheiro, R. R. (2013). Analysis of operating system diversity for intrusion tolerance. Software: Practice and Experience, doi: 10.1002/spe.2180 <http://dx.doi.org/10.1002/spe.2180>

SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2012; **00**:1–36
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe

Analysis of OS Diversity for Intrusion Tolerance

Miguel Garcia[1], Alysson Bessani[1], Ilir Gashi[2], Nuno Neves[1] and Rafael Obelheiro[3]

[1]*University of Lisbon, Faculty of Sciences*
[2]*City University London*
[3]*State of Santa Catarina University*

SUMMARY

One of the key benefits of using intrusion-tolerant systems is the possibility of ensuring correct behavior in the presence of attacks and intrusions. These security gains are directly dependent on the components exhibiting failure diversity. To what extent failure diversity is observed in practical deployment depends on how diverse are the components that constitute the system. In this paper we present a study with operating systems (OS) vulnerability data from the NIST National Vulnerability Database (NVD). We have analyzed the vulnerabilities of 11 different OSes over a period of 18 years, to check how many of these vulnerabilities occur in more than one OS. We found this number to be low for several combinations of OSes. Hence, although there are a few caveats on the use of NVD data to support definitive conclusions, our analysis shows that by selecting appropriate OSes one can preclude (or reduce substantially) common vulnerabilities from occurring in the replicas of the intrusion-tolerant system. 2012 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Diversity, Intrusion Tolerance, Operating Systems, Replication, Vulnerabilities.

1. INTRODUCTION

Many approaches are used by software developers and software system architects to minimize the risk of faults (bugs) or vulnerabilities[†] remaining in the code after a product is delivered for operational use. Examples of these approaches vary from code verification and validation techniques to various forms of software testing. However, all these tasks are usually performed with limited budgets and time constraints. Even if we assume that software producers employ good software development and testing practices, the complexity of the software systems, often built with several layers of off-the-shelf (OTS) components, make it very difficult to guarantee that (even modest) security requirements can be met. Given these constraints, software system architects succumb to the fact that systems will contain some faults and vulnerabilities even after release.

Fault or *intrusion tolerance* techniques then remain an interesting choice to ensure that a sufficiently dependable (or secure) service is delivered to the clients despite failures. A system is said to be intrusion-tolerant if it is able to remain functioning correctly even if some of its parts have

*Correspondence to: University of Lisbon, Faculty of Sciences, C6, Room 6.3.35, Campo Grande 1749-016 Lisbon, Portugal
email: mhenriques@lasige.di.fc.ul.pt phone: (+351) 217500532

[†]A specific type of bug, regarding security, is usually called a vulnerability. Once a vulnerability is discovered it can be maliciously exploited. If the exploited vulnerability leads to the software system deviating from its intended requirements or security policy then the system is deemed to have failed. The system can fail on a single or combination of the following security properties: confidentiality, availability and integrity. In the rest of this paper we will use the terms fault and vulnerability interchangeably.

their security compromised (for an overview of the area see [1]). A key building block of intrusion-tolerant systems is *Byzantine fault-tolerant* protocols, which guarantee correct behavior in spite of arbitrary faults, provided that a minority (usually less than one third [2]) of components is faulty. To satisfy this provision, system components need to exhibit failure diversity, i.e., the probability that a majority of components fail at the same time should be negligible (or else the system as a whole will fail). This failure diversity assumption is easier to justify when one is concerned with accidental faults, such as power outages, disk crashes, or message corruption due to noise in communication lines. However, for design faults of any kind, including security vulnerabilities, the assumption is difficult to guarantee. If multiple components contain the same vulnerabilities then a single attack can compromise all of them, therefore defeating the aim of intrusion tolerance system in providing improved security.

To reduce the probability of vulnerabilities existing in more than one component then *design diversity* [3] can be employed: each component uses diverse software to perform the same functions, with the expectation that the differences will reduce the occurrence of *common vulnerabilities*, i.e., vulnerabilities that exist in more than one system. Byzantine fault-tolerant replication often suggest the use of replica diversity (e.g., [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]), under the (sometimes implicit) assumption that they exhibit failure diversity. In this work we want to empirically assess to what extent failure diversity is exhibited in a complex category of off-the-shelf software, namely operating systems (OS).

We focus our study on OS because they are a category of software for which a myriad of available OTS options exist. Hence, system architects are more likely to reuse an available OTS OS rather than build their own. Given the variety of OSes available and the critical role that they play, diversity at this level may be a reasonable way of providing good security against common vulnerabilities at little extra cost. They also offer a good opportunity for diversity, since they are present in most computer systems, with many fully-featured, mature options available, even freely.

Hence, the main question we address in this paper is: *What are the security gains from using diverse OSes on a replicated intrusion-tolerant system?* To answer this question, we have studied the vulnerability data from the NIST National Vulnerability Database (NVD) [15] reported in the period between 1994 and 2011 for 11 operating systems distributions. This is the most complete vulnerability database available. The number of OS-related vulnerability reports in the NVD is sufficiently large to give meaningful results. Each vulnerability report in the NVD database contains (amongst other things) information about which products the vulnerability affects. We analyzed this dataset and checked how many vulnerabilities affect more than one operating system, i.e., common vulnerabilities. A reasonable pessimistic assumption is that, if a vulnerability reported in the NVD affects two or more OSes, then the same exploit (if it exists) can be used to compromise all affected OSes[‡]. In our study, as we will show, we found the number of common vulnerabilities to be relatively low for most pairs of operating systems.

A major distinctive of our work is the focus on common vulnerabilities. To explain better why this is important we will consider the interplay between the life cycle of vulnerabilities and the operational characteristics of intrusion-tolerant systems. Significant events in the vulnerability life cycle include [16]:

- *discovery*: when a vulnerability is first detected;
- *disclosure*: when the detector (the person who discovered or become aware of the vulnerability) first notifies the software vendor about his findings;
- *publication*: when the vulnerability is publicly disclosed; and
- *patching*: when a fix for the vulnerability becomes available.

Between discovery and patching, systems affected by a vulnerability can be compromised if an attacker succeeds in exploiting it (in many cases, it is possible to take steps to prevent exploitation in

[‡]This is a pessimistic assumption because a *common exploit* may not exist for these *common vulnerabilities*, i.e. more than one custom-made exploit may be needed for a common vulnerability, one for each OS in which the common vulnerability is found.

Prepared using speauth.cls
DOI: 10.1002/spe

unpatched systems, but there is still a window of vulnerability between discovery and publication). Attack tools that use vulnerabilities that have not been publicly disclosed are called 0-day exploits [17, 18]. Given that intrusion tolerance, due to the intrinsic cost of its associated mechanisms, is usually reserved for critical systems, it is reasonable to assume that the components of an intrusion-tolerant system will be fully patched, and cannot be compromised by publicly known vulnerabilities. Therefore, 0-day exploits pose the greatest danger, since there is little that can be done to protect against not-yet-known vulnerabilities. If such a vulnerability affects multiple components, there is a window of opportunity for compromising many, or even all, of them at the same time. Figure 1 illustrates an attack on a system designed to tolerate one fault (i.e., a $3f+1$ system): first the attacker needs to spend some time to discover a new vulnerability and craft a 0-day exploit for it; then she exploits that vulnerability on the first replica; and finally, without spending much more effort and time, she repeats the attack on at least f other replicas to compromise the system as a whole.

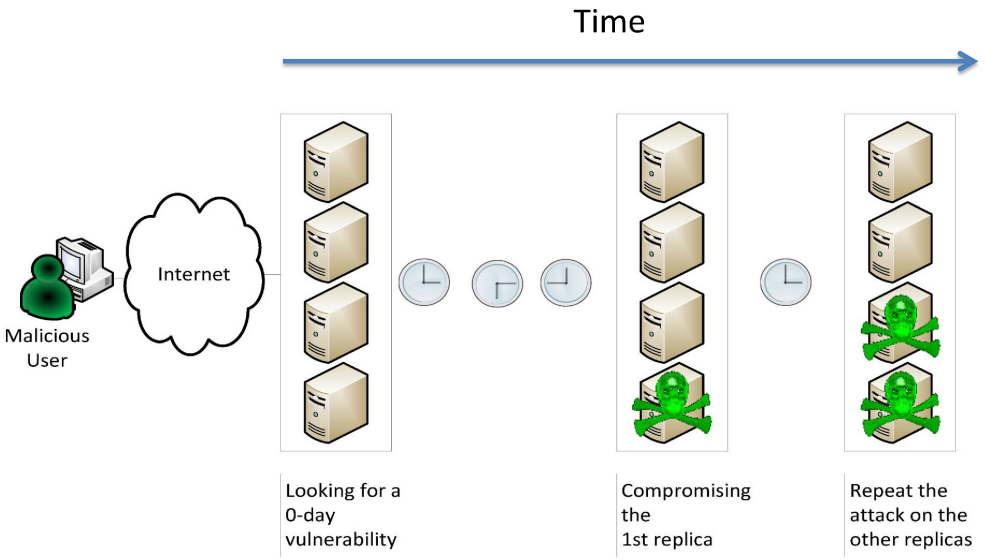

Figure 1. Example of an attack on a fully patched homogeneous replicated system.

The task of the attacker can be made more difficult by introducing diverse components on the replicas, rather than identical components. If the vulnerabilities that affect one replica do not appear on the others, a successful attack will require $f+1$ distinct 0-day vulnerabilities, one for each replica. This not only means that the attacker will have to spend more time and effort to penetrate the system, but also increases the likelihood that, by the time the last needed vulnerability is discovered, one of the others has already been patched. In any case, to what extent this component diversity leads to vulnerability diversity is difficult to answer except empirically.

The main contributions of our research can be summarized as follows:

1. A hand-made classification of the vulnerabilities that affect 11 operating systems in drivers, kernel, system software and applications;
2. A study of how many common vulnerabilities appear for several pairs of operating systems distributions divided in four OS families (BSD, Solaris, Linux and Windows) that capture different users' preferences;
3. Three strategies for selecting diverse software components in order to minimize the incidence of common vulnerabilities in replicated systems;
4. An analysis of common vulnerabilities across OS releases, assessing the degree to which different releases of an operating system show enough diversity to warrant its adoption as a simpler, less complex, more manageable configuration for replicated systems;
5. An in-depth discussion of the limitations and opportunities provided by the data available on NVD to assess the dependability and security properties of a system.

Prepared using speauth.cls
DOI: 10.1002/spe

Items 1, 2, and 5 have appeared previously in [19], but we have improved our discussion with updated data and further insights; items 3 and 4 are completely novel.

The rest of the paper is organized as follows. Section 2 presents an overview of related work on diversity, vulnerabilities on operating systems and studies using NVD data; Section 3 describes the methodology of the study, where we describe the NVD data source and the additional manual analysis we did to filter and enrich the dataset; Section 4 presents the main empirical analysis of the NVD data: Section 4.1 presents results from our classification of vulnerabilities in four types (kernel, drivers, system software and applications); in Section 4.2 we present the results on the number of vulnerabilities that affect more than one operating system; since we have analyzed the data over an 18-year period, we also looked at the evolution on the number of reported vulnerabilities for each OS family, and we present this analysis in Section 4.3. In Section 5 we present three strategies for selecting diverse operating systems using the data we have on common vulnerabilities; in the same section we also present some analysis of the potential benefits of diversity between releases of the same operating system, and consider the case where the system would be deployed with only two distinct OSes from the same family; in Section 6 we discuss the limitations we have found on NVD data and we reason about the employment of diversity in replicated systems. Finally in Section 7 we present conclusions and previsions for further work.

2. RELATED WORK

2.1. *Research on Diversity*

Design diversity was introduced in 1975 as a mechanism for software fault tolerance [20]. Randell advocated using spare components whose design was independent from the main components, in order to cope with the circumstances that caused failures in the latter. N-version programming is a technique for creating diverse software components introduced also in those early years [3]. The main idea behind this mechanism is to use N different implementations ("versions") of the same component, programmed by N different teams, ideally using distinct languages and development methodologies. Versions run concurrently with the same input, and form a consensus on the output from all versions to produce the final output. The objective is to achieve fault tolerance, assuming that designs and implementations developed independently will exhibit failure diversity. A later study by Knight and Leveson [21] showed that independently developed versions failed on the same demands with rates that were much higher than if statistical independence of failures between versions is assumed. Nevertheless the data from the same study [21] showed that the number of coincident failures was relatively low and that diversity can bring benefits in reducing overall system failure rates, even if statistical independence of failure rates between different versions cannot be safely assumed *a priori*. Hatton [22] corroborates this work by comparing the failure probability of a single version and a N-version solution.

The seminal work on using diversity to improve security is due to Joseph and Avizienis [23]. They advocate using diverse compilers (produced with N-version programming) to detect and mask Trojaned compilers that infect the generated executables with viruses. Later, Forrest and colleagues applied notions from biologic systems to computer security and argued that diversity is an important natural mechanism to reduce the effects of attacks [24, 25]. They proposed randomized compilation techniques that automatically create diversity in applications, but only implemented a stack layout randomizer that is effective against buffer overflow attacks. Taxonomies of diversity techniques for improving security have been introduced in [26, 27]. In the early 2000s, a report by noted security experts [28] denouncing the risks of software monocultures and championing diversity as a means of improving security raised controversy in the security industry [29]. However, most of the cited studies lack empirical or statistical evaluation on the effectiveness of diversity.

Experimental evidence of the benefits of adopting diversity of SQL database servers is presented in [30]. The authors analyzed bug reports for four database servers (PostgreSQL, Interbase, Oracle, and Microsoft SQL Server) and verified which products were affected by each bug reported (the focus of their study is on overall dependability, not specifically on security). They found a few

DOI: 10.1002/spe

cases of a single bug affecting more than one server, and that there were no coincident failures in more than two of the servers. Their conclusion is that diversity of off-the-shelf database servers is an effective means of improving system reliability. Some of the limitations of our data set (see Section 6) prevent us from making the same type of study with NVD data.

Littlewood and colleagues [31] survey a number of issues in software diversity modeling, presenting models that have been developed for assessing the reliability of systems that adopt diversity. The models discussed in the survey aim to provide a measure of the reliability of a system as a function of the demands presented to the system and how these demands influence the correctness of the behavior of the system; these parameters are, for the most part, expressed as probability distributions. Some of these ideas have later been extended to the security domain as well [32]. They show that, although failure independence cannot be claimed *a priori* when using diversity, diversity is nevertheless an effective means of increasing overall system reliability. They also discuss a number of caveats regarding software diversity modeling. It would be desirable to use these models in our context, but this is currently unfeasible, since we lack sufficiently detailed data (operational profiles and vulnerability exploitation rates) to apply them.

A study on diversity across software versions is presented in [33]. The authors propose a new discovery model that takes into account the software versions, and therefore the importance of shared code on vulnerability discovery. They used several versions of Apache and MySQL, two open source server applications, to cross-validate their model. It was found that vulnerabilities continue to be discovered for older versions due to code that is shared with newer, more popular versions.

The need for diversity in modern BFT/Survivable/Intrusion-tolerant systems was also explicitly explored in some works. Castro et al. [7] proposed BASE, a BFT replication framework based on PBFT [6], that uses off-the-shelf service components. This allows the utilization of different implementations to provide the same service, with the expectation of reducing the probability of common failures. Through the definition of the abstract state of the service being replicated the authors claim that it is possible to hide the distinct implementations of the services, exploiting opportunistic N-version programing with off-the-shelf software. They present an implementation of a BFT network file system with replicas using different UNIX-based OS and file systems. A similar experiment with heterogeneous replication was done by Distler et al. using a different replication framework to replicate a RUBiS middleware architecture [34]. Although these works support the use of diverse operating systems, they do not address the effectiveness of such approach in terms of dependability.

Roeder and Schneider [35] propose the use of proactive obfuscation, whereby each replica is periodically restarted using an automatically-generated diverse executable, by: address reordering and stack padding, system call reordering, instruction set randomization, heap and data randomization (e.g., a buffer overflow attack depends on stack layout, and therefore using entropy on the stack will likely crash the program instead of allowing an attacker to take control). The authors implemented two prototypes: 1) a distributed firewall based on pf (packet filter) in OpenBSD; and 2) a distributed storage service. Proactive obfuscation employs semantics-preserving program transformations. This approach makes the replicas' service diverse, but does not help with OS diversity, and thus is complementary to our study.

2.2. *Research on Bugs and Vulnerabilities in Operating Systems*

Given the criticality of operating systems, there are many papers that study the distribution of bugs and vulnerabilities in OS code. Miller et al. [36, 37] analyzed how commands and services in different UNIX variants dealt with random input and found out that between 25 and 50% of them (depending on the study) would crash or hang (seemingly enter an infinite loop). They identified cases where the same UNIX command would crash/hang in one OS but behave correctly in another one, as well as cases where both UNIX variants failed, but did not discuss whether occurrences of the latter could be attributed to the same bugs affecting different systems or not (some of the variants used in their studies derive from the same codebase). A comparison of the robustness of 15 different POSIX-based operating systems is presented in [38]. This study was based on fault injection: combinations of valid and invalid parameters were supplied to often-used system calls

DOI: 10.1002/spe

and C library functions, and the effects of this on reliability (e.g., system crash, process hang/crash, wrong or no error code returned, etc.) were observed. The authors found out some commonalities among the systems studied, especially with respect to the common mode failures of C library functions. However, from the available data it is impossible to conclude whether there were specific bugs that affected more than one system (the paper only shows how many failures were observed for each system call in various degrees of severity, according to the effect produced for each failure). Still, their evidence indicates that, from a reliability standpoint, using different operating systems reduces the number of common failure modes.

Chou et al. [39] used compiler extensions to perform static analysis of the Linux and OpenBSD kernels; their study shows that device drivers exhibit more flaws than the rest of the kernel, and that some types of bugs in the Linux kernel take an average of 1.8 years before being fixed. Ozment and Schechter [40] studied how OpenBSD security evolved over time, using data from OpenBSD security advisories and the project's source code repository to conclude that many vulnerabilities are still found in legacy code, that bugs in security-related code are more likely to be vulnerabilities, and that the rate of vulnerability reports for OpenBSD is decreasing over time. Anbalagan and Vouk [41] analyzed vulnerabilities in Bugzilla and Fedora Linux and found out that 34% of the vulnerabilities are exploited before being disclosed. None of these papers attempted to analyze the occurrence of common vulnerabilities across different OSes.

2.3. Research that has Used NVD data

Some vulnerability discovery models, which attempt to forecast the amount of vulnerabilities found in software, have been proposed [42, 43, 44]. Alhazmi and Malaiya [45] investigate how well these models fit with vulnerability data from the NVD, and conclude that the vulnerability discovery process follows the same S-shaped curve of "traditional" software reliability growth models [46], which measure all defects found in a system (not only those that affect security). This conclusion is disputed in [47], where it is claimed that the number of vulnerabilities disclosed in the NVD grows linearly with time (this contrast might be due to methodological differences). These studies cross-validate our idea of using the NVD as a source of vulnerability data; however, they are more concerned in modeling how many vulnerabilities are found in specific software over its lifetime [45] and if there are significant differences between open- and closed-source software [47], while our focus is on assessing the degree of diversity between different operating systems. Ozment [16] points out some limitations of the NVD database, which we discuss further in Section 6.

Gorbenko and colleagues proposed using a diversity-aware configuration manager for deploying web services in a cloud [48]. The manager uses vulnerability information from sources such as NVD and Common Vulnerabilities and Exposures (CVE) [49] to compute, in real-time, vulnerability scores for the available software infrastructures,[§] which are diverse. The infrastructure with the less vulnerable score is then chosen and, if necessary (it is different from the infrastructure currently deployed), redeployed by the manager. An analysis using OS vulnerability data for 2010 published in the NVD shows that switching the OS in response to new vulnerabilities reduces the number of days that a web service is at risk (the window of time between vulnerability disclosure and the release of a remedial patch). While on the surface this analysis is similar to our study, there are some substantial differences. First, Gorbenko et al. do not consider replicated systems, they choose the best single configuration from a pool of diversified options; therefore, they are not explicitly concerned with common vulnerabilities, which is a major focus of our work. Second, the analysis considers only data from 2010, while our study is much more comprehensive, spanning 18 years (1994–2011); an analysis based on data from a single year may not be sufficient for obtaining sufficently accurate estimates of the benefits of diversity.

[§] An infrastructure is a set containing an operating system, a web server, an application server, and a database management server.

Prepared using speauth.cls
DOI: 10.1002/spe

2.4. Other Related Empirical Security Evaluations

A study by Ransbotham [50] assessed the risk of vulnerability exploitation in open- vs. closed-source software using log data from intrusion detection systems (IDSs) across 960 firms as evidence of exploitation, and concluded that open-source is more attacked than closed-source, which was attributed to the availability of source code. However, not only IDS log data is noisy, potentially containing many false positives and missing false negatives, but this kind of data is not usually available to the public (Ransbotham used anonymized customer data provided by a managed security services company).

Rajnovic [51] analyzed data from 2356 US-CERT Vulnerability Notes (VNs) published between 2000 and 2010 and found out that 21% of these 2356 vulnerabilities affected more than one vendor, and that, on average, four vulnerabilities affecting multiple vendors were reported each month. His conclusion is that deploying diversity does not automatically ensure that there will be no common points of failure. It is important to note, however, that his analysis takes into consideration vendors, and not products, and is not restricted to operating systems. As a consequence, it is possible that several vulnerabilities accounted for in his study affect applications or other components that have little impact on servers, which are the focus of our study. It should also be observed that the NVD database is much larger than the US-CERT vulnerability notes database (in the same time frame of Rajnovic's study, the NVD records 42,292 vulnerabilities, roughly 18 times the number of VNs), which might influence the statistical conclusions.

3. METHODOLOGY

This section presents the methodology adopted in our study, with particular focus on how the data set (i.e., the vulnerabilities) was selected, processed and analyzed.

Data source. We have analyzed OS vulnerability data from the NVD database [15]. NVD uses the CVE definition of vulnerability [52], which is given below:

> *Definition 1* (CVE Vulnerability)
> 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.
>
> CVE considers a mistake a vulnerability if it allows an attacker to use it to violate a reasonable security policy for that system (this excludes entirely "open" security policies in which all users are trusted, or where there is no consideration of risk to the system).
>
> For CVE, a vulnerability is a state in a computing system (or set of systems) that either:
>
> - 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.

NVD aggregates vulnerability reports from more than 70 security companies, forums, advisory groups and organizations[¶], being thus the most complete vulnerability database on the web. All data is made available as XML files containing the reported vulnerabilities on a given period, called *data feeds*. We analyze feeds from 2002 to 2011[‖].

[¶]See the complete list on http://cve.mitre.org/compatible/alerts_announcements.html.
[‖]The 2002 feed includes information about vulnerabilities that were reported between 1994 and 2002. The most recent feed that was analyzed in this paper contained vulnerabilities until the end of 2011.

Prepared using speauth.cls
DOI: 10.1002/spe

Each NVD data feed contains a list of reported vulnerabilities sorted by its date of publication on a given period. For each vulnerability, called *entry* in the NVD parlance, interesting information is provided such as: a unique name for the entry, in the format CVE-*YEAR-NUMBER*; the list of products (with version numbers) affected by the vulnerability; the date of vulnerability publication; and the security attributes that are affected when the vulnerability is exploited on a system.

We developed a program that collects, parses and inserts the XML data feeds into an SQL database, deployed with a custom schema to group vulnerabilities by affected products and versions.

Data selection. Despite the large amount of information about each vulnerability available in NVD, for the purposes of this study we are only interested in the name, publication date, summary (description), type of exploit (local or remote), and list of affected configurations. We have collected vulnerabilities reported for 64 Common Platform Enumerations (CPEs) [53]. Each one of these describes a system, i.e., a stack of software/hardware components in which the vulnerability may be exploited. These CPEs were filtered, resulting in the following information that was stored in our database:

- **Part:** NVD separates this in Hardware, Operating System and Application. For the purpose of this study we choose only enumerations marked as Operating System;
- **Product:** The product name of the platform;
- **Vendor:** Name of the supplier or vendor of the product platform.

Those 64 CPEs were, by manual analysis, clustered in 11 server-based OS distributions: *OpenBSD, NetBSD, FreeBSD, OpenSolaris, Solaris, Debian, Ubuntu, Red Hat*[**], *Windows2000, Windows2003* and *Windows2008*. These distributions cover the most used *server OS products* of the BSD, Solaris, Linux and Windows families (Mac OS X was not included due to its low popularity on the server market – see, e.g., [54]).

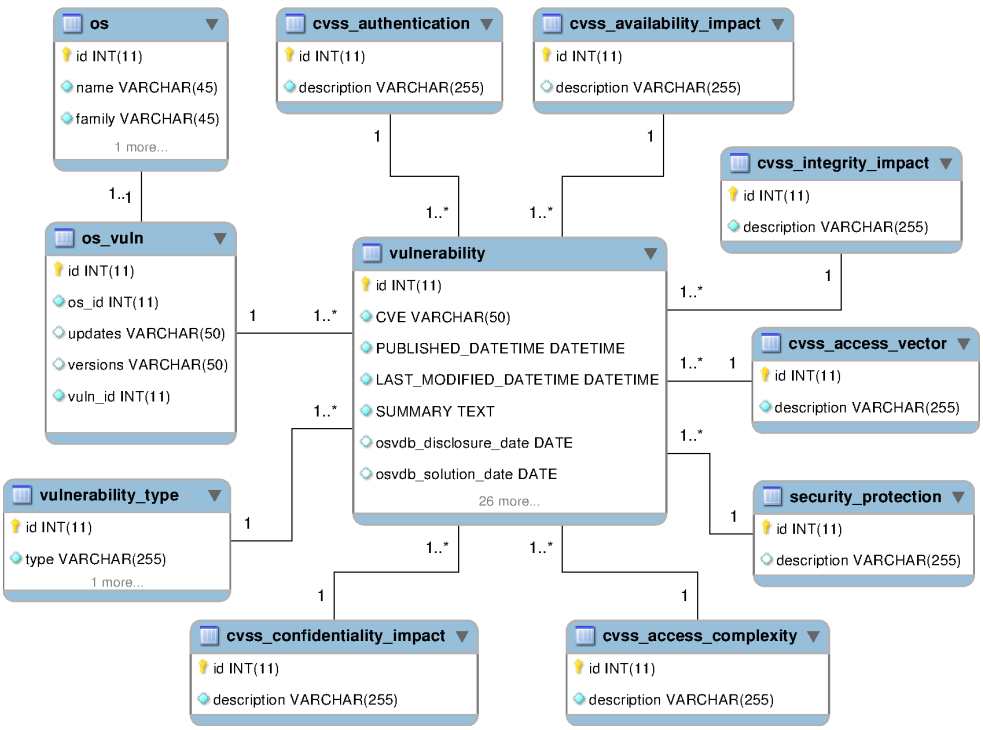

Figure 2. Simplified SQL schema of the database used to store and analyze the NVD data. Fields not displayed are only used for auxiliary purposes.

[**] *Red Hat* comprises the "old" Red Hat Linux (discontinued in 2003) and the newer Red Hat Enterprise Linux (RHEL).

The schema of the resulting database is displayed in Figure 2. The most important tables are:

- *cvss_** tables: refer directly to the Common Vulnerability Scoring System (CVSS) metrics [55] of the stored vulnerabilities, which quantify the severity and impact of vulnerabilities;
- *vulnerability*: stores basic data about a vulnerability (name, publication date, etc.) from the NVD augmented with vulnerability life cycle information from the Open Source Vulnerability Database (OSVDB) [56];
- *vulnerability_type*: stores the vulnerability type assigned by us (see Section 4.1);
- *os*: stores the operating systems platforms of interest in this study;
- *os_vuln*: stores the relationship between vulnerabilities and operating systems, and their affected versions.

The use of an SQL database brings at least three benefits when compared with analyzing the data directly from the XML feeds. First, it allows us to *enrich the data set* by hand, for example, by assigning, to each vulnerability, information regarding its type (see Section 4.1), and also by associating release times and family names to each affected OS distribution. Second, it allows us to modify the CVE fields to correct problems. For instance, one of the problems with NVD is that the same product is occasionally registered with distinct names in different entries; e.g., (*debian_linux*, *debian*) and (*linux*, *debian*) are two (product, vendor) pairs we have found for the Debian Linux distribution. This same problem was previously observed by other users of NVD data feeds [57]. Finally, an SQL database is much more convenient to work with than parsing the feeds on demand.

3.1. Filtering the Data

From the more than 44,000 vulnerabilities published by NVD at the time of this study, we selected 2563 vulnerabilities. These vulnerabilities are the ones classified as OS-level vulnerabilities ("/o" in their CPE) for the operating systems under consideration.

When manually inspecting the data set, we discovered and removed vulnerabilities that contained tags in their descriptions such as *Unknown* and *Unspecified*. These correspond to vulnerabilities for which NVD does not know exactly where they occur or why they exist (however, they are usually included in the NVD database because they were mentioned in some patch released by a vendor). We also found few vulnerabilities flagged as ***DISPUTED***, meaning that product vendors disagree with the vulnerability existence, and *Duplicate*, used for vulnerabilities in which the *summary* points to a duplicate entry or duplicate entry suspicion. Due to the uncertainty that surrounds these vulnerabilities, we decided to exclude them from the study as well.

Table I shows the distribution of these vulnerabilities across the analyzed OSes, together with the total number of valid vulnerabilities.

Table I. Distribution of OS vulnerabilities in NVD.

OS	Valid	Unknown	Unspecified	Disputed	Duplicate
OpenBSD	153	1	1	1	0
NetBSD	143	0	1	2	0
FreeBSD	279	0	0	2	0
OpenSolaris	31	0	52	0	0
Solaris	426	40	145	0	3
Debian	213	3	1	0	0
Ubuntu	90	2	1	0	0
Red Hat	444	13	8	1	1
Windows2000	495	7	28	5	5
Windows2003	56	5	34	3	5
Windows2008	334	0	8	0	3
#distinct vulns.	2270	63	210	8	12

An important observation about Table I is that the columns do not add up to the number of distinct vulnerabilities (last row of the table) because some vulnerabilities are shared among OSes

Prepared using speauth.cls
DOI: 10.1002/spe

and are counted only once. Notice that about 60% of the removed vulnerabilities affected Solaris and OpenSolaris. Moreover, these two systems are the only ones that have more than 10% of its vulnerabilities removed. We should remark that this manual filtering was necessary to increase the confidence that only valid vulnerabilities were used in the study.

4. OS DIVERSITY STUDY

This section describes the results of the study. It provides an overall analysis of the counts of vulnerabilities in each OS component class, presents vulnerabilities that are common to OS pairs, and shows the evolution of shared OS vulnerabilities over time. The section also gives empirical evidence to demonstrate that there are security gains in using diverse OSes in a replicated system.

4.1. Vulnerability Distribution by OS Component

Nowadays, when users acquire or download an operating system, they get a kernel implementing the fundamental OS functionality but also a number of other software components. After the initial installation, users may reconfigure the machine to their needs, for instance by removing application packages that are unnecessary and/or by including device drivers to support new hardware. This customization is also advisable from a security standpoint, with a few different baseline OS installations created for machines with distinct roles within an organization. For example, a network server OS installation should be stripped of most applications to minimize the attack surface and reduce the risk of intrusion. Therefore, besides understanding how vulnerabilities affect the various operating system products, it is also important to know what parts of these systems are compromised by the vulnerabilities. In NVD, an *operating system product* is composed by the kernel, device drivers, optional modules, system software and applications. Since NVD does not have a specific field identifying the affected component except for the vulnerability description, we inspected the entries manually and classified each vulnerability in one of four categories: *Driver, Kernel, System Software* and *Application*. The classification criteria are the following:

- **Kernel:** vulnerabilities in the TCP/IP stack (and other network protocols implemented in the OS), file system, process and task management, core libraries and modules related to the CPU architecture;
- **Driver:** vulnerabilities in drivers for the devices that might be connected to the machine (for example, wireless/wired network cards, video/graphic cards, web cams, audio cards, plug and play devices);
- **System Software:** vulnerabilities in software providing common OS functionalities such as login, shells and basic daemons (e.g., DNS, telnet, lpd). In this class, we only account for software that comes by default with the distribution (although sometimes it is possible to uninstall these components without affecting the main OS operation);
- **Application:** vulnerabilities in software that comes with the OS but that is not needed for basic operation, and in some cases requires specific installation. For example, database management systems, messaging clients, text editors and processors, web/email/FTP clients and servers, music/video players, programming languages (compilers and virtual machines), security packages (antivirus and authentication solutions), and games.

Table II summarizes the result of the analysis of the 2270 vulnerability descriptions, which were then assigned to one of the OS component classes. The table shows that, with the exception of *Drivers*, all OS products have a reasonable number of vulnerabilities in each class. Therefore, this confirms the intuition that there are security gains if certain software components can be removed from the OS distributions. In the BSD and Solaris families, vulnerabilities appear in high numbers in the *Kernel* part, while in the Linux and Windows families, the *Applications* vulnerabilities are more prevalent. This can be explained by noticing that Windows and Linux default distributions usually contain more pre-installed applications when compared to BSD/Solaris. Therefore, the number of *Applications* vulnerabilities in Windows and Linux have higher reported values – there is the

Prepared using speauth.cls
DOI: 10.1002/spe

expectation that users will not install most of the applications missing from the default distributions, and thus their vulnerabilities not appear in the OS statistics.

Table II. Vulnerabilities per OS component class.

OS	Driver	Kernel	Sys. Soft.	Application	Total
OpenBSD	2	76	37	38	153
NetBSD	9	64	39	31	143
FreeBSD	4	153	61	61	279
OpenSolaris	0	15	9	7	31
Solaris	2	155	120	149	426
Debian	1	25	39	148	213
Ubuntu	2	22	8	58	90
Red Hat	5	94	108	237	444
Windows2000	3	146	135	211	495
Windows2003	2	171	96	291	560
Windows2008	0	123	36	175	334
% Total	1.0%	33.5%	22.5%	42.9%	

The last row of the table presents the percentage of each class on the total data set. One can observe that most vulnerabilities occur in the Application and Kernel components, which is then followed by the System Software group of utility programs. It is interesting to notice that Drivers have a very small percentage of the published OS vulnerabilities. This result is somewhat surprising because device drivers account for a large fraction of OS code [39], and since they are typically hard to develop, one would expect that they would contain a large number of programming flaws. In fact, previous studies have demonstrated that drivers are the major cause for crashes in certain OSes [58]. One, however, should keep in mind that software bugs do not necessarily translate into security vulnerabilities, which are the kind of flaw reported in the NVD. In order to become vulnerabilities, the adversary has to be able to force the conditions to activate the bugs, which for device drivers can be extremely difficult because they are related to specific low-level aspects of the system.

4.2. Common Vulnerabilities

This section analyzes the vulnerabilities that were shared in OS pairs over the period of 1994 to 2011. In the investigation, we consider three possible server machine setups offering an increasingly more secure platform. This accommodates the case where system administrators create differentiated OS installations, which contain more or less vulnerabilities depending on the services and applications available in the server. Of course, other setups could be used, but we decided to concentrate on these configurations because they are quite generic and they lead to results that can be directly obtained from the NVD data. The setups are the following:

- **Fat Server:** The server contains most of the software packages for a given OS, and consequently, it can be used to run various kinds of applications by locally or remotely connected users. This server has potentially all vulnerabilities that were reported for the corresponding OS;
- **Thin Server:** This setting corresponds to a platform that does not contain any applications (with the exception of the replicated service). The server has a decreased security risk because the attack vectors related to applications have been mostly eliminated. In this setting, vulnerabilities classified as *Applications* are excluded from the statistics (see Section 4.1);
- **Isolated Thin Server:** The server is configured similarly to the Thin Server but is placed in a room physically protected from illegal accesses, with remote logins disabled, and therefore it can only be compromised by receiving malicious packets from the network. In this setting, we only consider remotely exploitable vulnerabilities (those with "Network" or "Adjacent Network" values in their CVSS_ACCESS_VECTOR field) that are not classified as *Applications*.

Prepared using speauth.cls
DOI: 10.1002/spe

The Fat Server setup corresponds to a case where any application available in the OS distribution may be targeted by attack. Therefore, it provides an upper bound estimate on the number of vulnerabilities that can be exploited. The Thin Server setups (Isolated or not) are the counterpart case, where the system is deployed without applications that come bundled with the OS, and thus provide a lower bound estimate on the number of common vulnerabilities. The Isolated Thin Server, in particular, only has the vulnerabilities that can be remotely exploited. Of course, in practice, at least one application (such as a name service or a distributed file service) would be deployed in an intrusion-tolerant system, and the vulnerabilities of this application would add up to the flaws that we will report. In any case, since we expect that most of the complexity is in the operating system, we anticipate that a single application will have a small contribution to the overall number of vulnerabilities.

Table III shows the vulnerabilities that were found in common for every combination of OS pair. In all cases, the number of shared vulnerabilities between two OSes is substantially reduced when compared to the overall set of vulnerabilities. Even considering a Fat Server configuration, it is possible to find out OS pairs that do not have common flaws (e.g., NetBSD-Ubuntu), and OS families that share very few vulnerabilities (e.g., BSD and Windows). As expected, OSes from the same family have more common vulnerabilities due to the software components and applications that are reused (e.g., Debian-Red Hat or Windows2000-Windows2003). The Thin Server when compared with the Fat Server shows improvements in several OS pairs, but often there are still a few shared flaws. In contrast, the use of an Isolated Thin Server has a much stronger impact on security because it substantially diminishes the number of common vulnerabilities – the number of pairs with zero vulnerabilities goes from 11 to 21. Overall, this means that a significant portion of common vulnerabilities are local (i.e., cannot be exploited remotely) and/or come from applications that are available in both operating systems.

Table IV shows which parts of the OS are affected by common vulnerabilities in an Isolated Thin Server configuration, considering only the OS pairs with non-zero common vulnerabilities. The fact that there are many shared *Kernel* and *System Software* vulnerabilities between Windows2000 and Windows2003 indicates that the latter inherits considerable parts of the OS from its predecessor. This same trend is also observed between Windows2008 and Windows2003/Windows2000, although to a less extent. Interestingly, no single vulnerable driver is present in all products, which can be explained by the very few faulty drivers that are reported.

The second OS family with more common vulnerabilities is BSD, which also reuses several OS components. Here, a few vulnerabilities were found in device drivers that appear in more than one OS pair. A somewhat surprising result is the fact that most Linux distributions have much less shared flaws than we anticipated. We inspected the vulnerabilities manually in order to find an explanation, and we discovered that Linux distributions customize both their kernel and system software, and thus the vulnerabilities are less common. Another interesting point about OSes from the Linux family is that they have an almost negligible number of driver vulnerabilities (see Table II), and none of them appears in more than one OS.

OpenSolaris is an open source operating system based on Solaris that was released in 2008, and until this moment the two OSes only share a small number of vulnerabilities.

Intrusion-tolerant systems are usually built using four or more replicated servers. Therefore, it is useful to understand if there are many vulnerabilities that involve groups of OSes larger than two – if this is the case, then it will be hard to find OS configurations for the various servers that do not share vulnerabilities, and a goal of using off-the-shelf diversity will be difficult to be achieved in practice. Figure 3 portrays the number of common vulnerabilities that exist simultaneously in increasingly larger sets of operating systems (with Isolated Thin Servers). The graph shows a rapid decrease of the shared vulnerabilities as the number of OSes grows. The largest group that was affected by the same vulnerability had six OSes, and this occurred only for a single flaw, and there were also two vulnerabitlities in sets with five OSes. Vulnerabilities in two and three OSes normally occur in systems from the same family, whose common ancestry implies the reuse of larger portions of the code base. As we have seen in previous tables, this is particularly true for the Windows and BSD families.

Prepared using speauth.cls
DOI: 10.1002/spe

Table III. Vulnerabilities (1994 to 2011): *Fat Server* - all vulnerabilities; *Thin Server* - no application vulnerabilities; *Isolated Thin Server* - no application vulnerabilities and only remotely-exploitable vulnerabilities. The columns with *v(A)* and *v(B)* show the total number of vulnerabilities collected for OSes A and B respectively, whereas *v(A,B)* is the count of vulnerabilities that affect both the A and B systems.

Operating Systems	**Fat Server**			**Thin Server**			**Isolated Thin Server**		
Pairs (A-B)	$v(A)$	$v(B)$	$v(A,B)$	$v(A)$	$v(B)$	$v(A,B)$	$v(A)$	$v(B)$	$v(A,B)$
OpenBSD-NetBSD	153	143	45	115	112	34	62	46	17
OpenBSD-FreeBSD		279	57		218	49		90	33
OpenBSD-OpenSolaris		31	1		24	1		6	0
OpenBSD-Solaris		426	13		277	10		108	6
OpenBSD-Debian		213	2		65	2		28	0
OpenBSD-Ubuntu		90	3		32	1		10	0
OpenBSD-Red Hat		444	11		207	6		69	4
OpenBSD-Win2000		495	3		284	3		183	3
OpenBSD-Win2003		560	2		269	2		138	2
OpenBSD-Win2008		334	1		159	1		56	1
NetBSD-FreeBSD	143	279	54	112	218	41	46	90	25
NetBSD-OpenSolaris		31	0		24	0		6	0
NetBSD-Solaris		426	18		277	14		108	10
NetBSD-Debian		213	5		65	4		28	4
NetBSD-Ubuntu		90	0		32	0		10	0
NetBSD-Red Hat		444	12		207	9		69	6
NetBSD-Win2000		495	3		284	3		183	3
NetBSD-Win2003		560	1		269	1		138	1
NetBSD-Win2008		334	1		159	1		56	1
FreeBSD-OpenSolaris	279	31	0	218	24	0	90	6	0
FreeBSD-Solaris		426	23		277	15		108	8
FreeBSD-Debian		213	7		65	4		28	1
FreeBSD-Ubuntu		90	3		32	3		10	0
FreeBSD-Red Hat		444	20		207	13		69	5
FreeBSD-Win2000		495	4		284	4		183	4
FreeBSD-Win2003		560	2		269	2		138	2
FreeBSD-Win2008		334	1		159	1		56	1
OpenSolaris-Solaris	31	426	27	24	277	22	6	108	6
OpenSolaris-Debian		213	1		65	1		28	0
OpenSolaris-Ubuntu		90	1		32	1		10	0
OpenSolaris-Red Hat		444	1		207	1		69	0
OpenSolaris-Win2000		495	0		284	0		183	0
OpenSolaris-Win2003		560	0		269	0		138	0
OpenSolaris-Win2008		334	0		159	0		56	0
Solaris-Debian	426	213	4	277	65	4	108	28	2
Solaris-Ubuntu		90	2		32	2		10	0
Solaris-Red Hat		444	17		207	10		69	6
Solaris-Win2000		495	10		284	3		183	3
Solaris-Win2003		560	8		269	1		138	1
Solaris-Win2008		334	1		159	0		56	0
Debian-Ubuntu	213	90	15	65	32	6	28	10	2
Debian-Red Hat		444	66		207	28		69	13
Debian-Win2000		495	1		284	1		183	1
Debian-Win2003		560	1		269	1		138	1
Debian-Win2008		334	0		159	0		56	0
Ubuntu-Red Hat	90	444	28	32	207	8	10	69	1
Ubuntu-Win2000		495	1		284	1		183	1
Ubuntu-Win2003		560	1		269	1		138	1
Ubuntu-Win2008		334	0		159	0		56	0
Red Hat-Win2000	444	495	2	207	284	1	69	183	1
Red Hat-Win2003		560	2		269	1		138	1
Red Hat-Win2008		334	0		159	0		56	0
Win2000-Win2003	495	560	265	284	269	120	183	138	85
Win2000-Win2008		334	80		159	29		56	16
Win2003-Win2008	560	334	282	269	159	125	138	56	40

DOI: 10.1002/spe

Table IV. Common vulnerabilities on Isolated Thin Servers (only OS pairs with non-zero shared flaws).

OS Pairs	*Driver*	*Kernel*	*Sys. Soft.*	*Total*
Win2000-Win2003	0	42	43	85
Win2003-Win2008	0	18	22	40
OpenBSD-FreeBSD	1	14	18	33
NetBSD-FreeBSD	2	13	10	25
OpenBSD-NetBSD	1	8	8	17
Win2000-Win2008	0	8	8	16
Debian-Red Hat	0	5	8	13
NetBSD-Solaris	0	4	6	10
FreeBSD-Solaris	0	5	3	8
OpenSolaris-Solaris	0	3	3	6
OpenBSD-Solaris	0	5	1	6
Solaris-Red Hat	0	3	3	6
NetBSD-Red Hat	0	0	6	6
FreeBSD-Red Hat	0	1	4	5
OpenBSD-Red Hat	0	1	3	4
NetBSD-Debian	0	0	4	4
FreeBSD-Win2000	1	3	0	4
OpenBSD-Win2000	0	3	0	3
NetBSD-Win2000	1	2	0	3
Solaris-Win2000	0	3	0	3
OpenBSD-Win2003	0	2	0	2
FreeBSD-Win2003	0	2	0	2
Solaris-Debian	0	1	1	2
Debian-Ubuntu	0	0	2	2
NetBSD-Win2003	0	1	0	1
NetBSD-Win2008	0	1	0	1
FreeBSD-Debian	0	0	1	1
FreeBSD-Win2008	0	1	0	1
Debian-Win2000	0	0	1	1
Ubuntu-Red Hat	0	0	1	1
Ubuntu-Win2000	0	0	1	1
Red Hat-Win2000	0	0	1	1
OpenBSD-Win2008	0	1	0	1
Solaris-Win2003	0	1	0	1
Debian-Win2003	0	0	1	1
Ubuntu-Win2003	0	0	1	1
Red Hat-Win2003	0	0	1	1

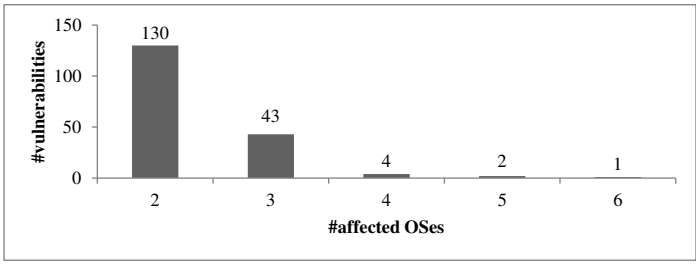

Figure 3. Common vulnerabilities for *n* different OSes (with Isolated Thin Servers).

Table V lists in more detail the vulnerabilities that can be exploited in larger groups (four to six) of OSes. The first three bugs have a considerable impact because they allow a remote adversary to run arbitrary commands on the local system using a high-privilege account. They occurred either in widespread login services (telnet and rlogin) or in a basic system function, and consequently several products from the BSD family were affected, as well as Solaris. The NVD entry for the CVE-2001-0554 vulnerability also had external references to the Debian and Red Hat websites, which

DOI: 10.1002/spe

could indicate that these systems might suffer from a similar (or the same) problem. Vulnerability CVE-2008-1447 occurs in a large number of systems because it results from a bug in the BIND implementation of the Domain Name System (DNS). Since BIND is a highly popular service, more OSes could potentially be affected. A closer look at the corresponding NVD entry reveals external references to the sites of OpenBSD, NetBSD, and FreeBSD, indicating that they would be vulnerable in case this software was being used. This sort of vulnerability confirms that from an intrusion-tolerance perspective it is unwise to run the same server software everywhere, and that diverse implementations must be selected (in this case for recursive DNS servers).

Table V. Vulnerabilities that affect more than four OSes.

CVE	#affected OSes	Description
CVE-1999-0046	4	Buffer Overflow of *rlogin* allows admin access. Affects: NetBSD, FreeBSD, Solaris and Debian.
CVE-2001-0554	4	Buffer overflow in telnetd (telnet daemon) allows remote attackers to execute arbitrary commands. Affects: OpenBSD, NetBSD, FreeBSD and Solaris.
CVE-2003-0466	4	Off-by-one error in the Kernel function *fb_realpath()* allows admin access. Affects: OpenBSD, NetBSD, FreeBSD and Solaris.
CVE-2005-0356	4	TCP implementations allow a denial of service (DoS) via spoofed packets with large timer values, when used with Protection Against Wrapped Sequence Numbers. Affects: OpenBSD, FreeBSD, Windows2000 and Windows2003.
CVE-2008-1447	5	The BIND 8 and 9 implementation of the DNS protocol allow attackers to spoof DNS traffic via a birthday attack to conduct cache poisoning against recursive resolvers. Affects: Debian, Ubuntu, Red Hat, Windows2000 and Windows2003.
CVE-2001-1244	5	TCP implementations allow a DoS by setting a maximum segment size very small to force the generation of more packets, amplifying network traffic and CPU consumption. Affects: OpenBSD, NetBSD, FreeBSD, Solaris and Windows2000.
CVE-2008-4609	6	TCP implementation allows a DoS via multiple vectors that manipulate TCP state table. Affects: OpenBSD, NetBSD, FreeBSD, Windows2000, Windows2003 and Windows2008.

The remaining three vulnerabilities are related to the TCP/IP protocol stack. All of them affect system availability, allowing different forms of denial of service (DoS) attacks. CVE-2008-4609 is the vulnerability that affects more operating systems, according to the NVD database. Given that TCP/IP stack code is often reused across operating systems, we checked the web sites of the other OSes for reports related to this vulnerability. We only found a disclaimer by Red Hat [59] stating that this flaw affected some releases but that they would not provide an update (they only offered a mitigation solution based on IPtables, the Linux firewall software).

Overall, the above results look encouraging because over a large period of time (around 18 years) there are very few vulnerabilities that appear in many operating systems. A good portion of them are in the TCP/IP stack implementation, which is probably the most shared software component across OSes, but they only impact on the availability of the system, leaving confidentiality and integrity of data unharmed.

4.3. Evolution of Common Vulnerabilities over Time

This section studies how OS vulnerabilities evolved between the years 2000 and 2011. Figure 4 depicts the total number of reported vulnerabilities for each OS in a Fat Server configuration. Since certain OSes only started to be released after 2000, some graphs have a few years at the beginning without vulnerabilities: Windows2003 was first released in 2003, Ubuntu in 2004, and Windows2008 and OpenSolaris in 2008. We also do not present any values in the last column of the Windows2000 and OpenSolaris graphs, since these OSes were not included in the NVD database in

2011. This occurred probably because in 2010 Microsoft ended support for Windows2000 [60], and the future of OpenSolaris became uncertain with the acquisition of Sun by Oracle [61].

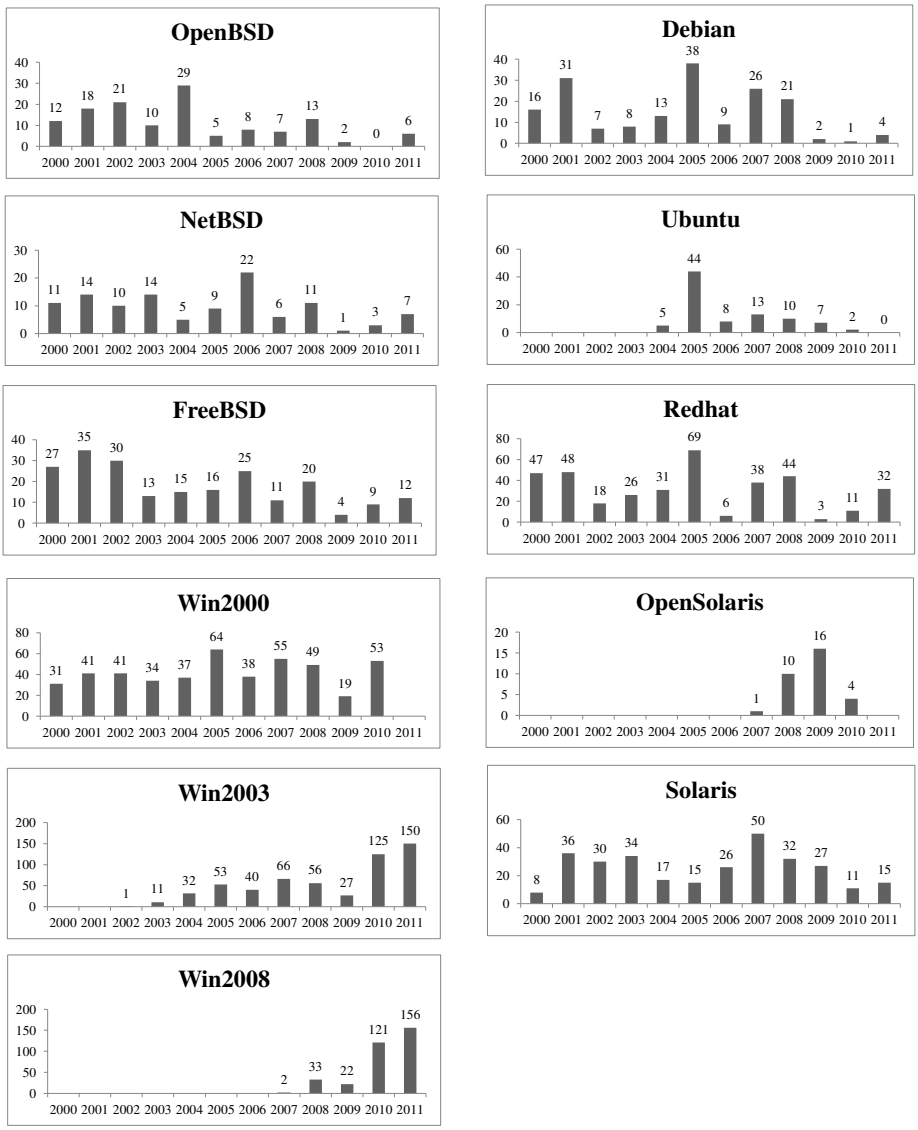

Figure 4. Number of vulnerabilities per year (with Fat Servers).

The way vulnerabilities evolved over the 12-year period allows the following observations. First, it is possible to notice a reasonable correlation among the peaks and valleys of both the Windows and Linux families, and, to a lesser extent, in the BSD family. This supports the idea that some vulnerabilities are shared across the family members, as seen in the previous section. Second, some OS families have less vulnerabilities being reported in recent years (2009–2011) than in the more distant past. This is true for both the BSD and Linux, which could indicate that the systems are becoming more stable, but also that the development processes being employed impose stronger requirements on software quality. Finally, it is also useful to compare the vulnerability dates and the year of the first OS release. NVD classifies vulnerabilities when they are first discovered, and then lists the OSes that might be compromised by their exploitation. Therefore, it is possible to find for example in OpenSolaris one entry earlier than the official release date of 2008, a vulnerability shared with Solaris (a locally exploitable bug – CVE-2007-5365). This confirms that OpenSolaris inherits some of the code of Solaris, and consequently, it is influenced by existing flaws. The same

Prepared using speauth.cls
DOI: 10.1002/spe

type of problem was also observed with Windows 2000 with regard to vulnerabilities in common with Windows NT[††].

Table VI displays the number of common vulnerabilities per year in OS pairs in a Fat Server setting. We have excluded OpenSolaris from this analysis because data is missing for several years over the period. The table displays a significant number of OS pairs with zero common vulnerabilities in many years (45% of the non-empty cells have zeros). This is particularly visible for OSes belonging to different families, such as between Debian and Solaris, or Debian and Windows2008. Years with zero entries even appear for pairs with very high vulnerability counts, as bugs sometimes cluster around certain years, instead of being evenly distributed (e.g., Debian-Red Hat). With the exception of the OSes within the Windows family, which usually share many flaws, the maximum number of vulnerabilities was 20 for Debian-Red Hat in 2005. In the remaining cases the numbers were much smaller, with only three other OS pairs above 10 (Ubuntu-Red Hat in 2005, Debian-Red Hat in 2001, and OpenBSD-FreeBSD in 2002). Since an intrusion-tolerant system might be used only for a limited amount of time, such as during a temporary event (e.g., the Olympic Games or the World Cup), the table results support the idea that many possible OS configurations can be found without common vulnerabilities over short intervals. Even for one-year periods, it is possible to find cases where most OS pairs have zero common vulnerabilities (e.g., 2011, 2010, 2009 and 2006). These observations are even more apparent in an Isolated Thin Server configuration, since the relevant vulnerabilities are significantly reduced (see Table III). Here, the percentage of zero cells is even higher, reaching the value of 63%.

The last rows of the table display the mean, standard deviation and maximum number of vulnerabilities for each year. The mean value increases in the most recent years, implying that more shared vulnerabilities are being reported on average. This however should not be understood as an overall decrease in diversity across systems. As the standard deviation demonstrates, there are a few OSes with a very high number of common flaws (the Windows family), while for the rest the number of common vulnerabilities is decreasing. In fact, if the influence of Windows2008 is removed from the table, the mean number of common vulnerabilities has been reduced in the last three years.

From the point of view of deploying intrusion-tolerant systems, the results of this section show that there are several OS pairs with a few to none shared vulnerabilities over reasonable time intervals (a couple of years). Moreover, in certain cases, these vulnerabilities seem to be decreasing as systems mature. Consequently, it should be feasible to select configurations of four or more operating systems that with high probability are very resilient to intrusions.

5. STRATEGIES FOR OS DIVERSITY SELECTION

This section presents three alternative strategies to deploy diverse operating systems on replicated systems, to decrease the chance of common vulnerabilities. We start the section by first describing an approach we called the Common Vulnerability Indicator (CVI), which is used by one of the strategies. For each strategy, we then give example sets of OSes that exhibit a good level of diversity, and perform an evaluation based on the collected data. Finally we conclude the section with an analysis of potential diversity between releases of the same OSes (concentrating on the OSes that the three strategies picked as the best configuration).

5.1. Common Vulnerability Indicator

We already saw in the previous section that the number of common vulnerabilities that are observed between different operating systems varies over time. In order to be able to evaluate which OS pairs are more (or less) likely to experience common flaws while taking into account the timing of

[††]We found three cases in other OS versions where a vulnerability was reported much earlier than the corresponding release. After examining the NVD entry, we were able to exclude them as errors in the database, and therefore, they are not shown in the graphs.

DOI: 10.1002/spe

Table VI. Number of common vulnerabilities for OS pairs between 2000 and 2011 (with Fat Servers).

	2000	01	02	03	04	05	06	07	08	09	10	2011
OpenBSD-NetBSD	5	7	5	2	2	0	3	2	7	0	0	4
OpenBSD-FreeBSD	2	6	11	6	6	2	2	1	7	0	0	4
OpenBSD-Solaris	0	2	3	2	1	0	0	1	0	0	0	1
OpenBSD-Debian	0	0	0	0	0	0	0	1	1	0	0	0
OpenBSD-Ubuntu	-	-	-	-	0	0	0	3	0	0	0	0
OpenBSD-Red Hat	3	0	1	0	3	0	0	3	0	0	0	0
OpenBSD-Win2000	0	1	0	0	0	1	0	0	1	0	0	-
OpenBSD-Win2003	-	-	-	0	0	1	0	0	1	0	0	0
OpenBSD-Win2008	-	-	-	-	-	-	-	-	1	0	0	0
NetBSD-FreeBSD	4	6	6	5	1	0	3	1	9	1	2	4
NetBSD-Solaris	0	2	0	4	0	0	2	0	0	1	0	1
NetBSD-Debian	0	2	2	0	0	0	0	0	0	0	0	0
NetBSD-Ubuntu	-	-	-	-	0	0	0	0	0	0	0	0
NetBSD-Red Hat	2	2	2	0	0	0	0	0	0	0	0	0
NetBSD-Win2000	0	1	0	1	0	0	0	0	1	0	0	-
NetBSD-Win2003	-	-	-	0	0	1	0	0	1	0	0	0
NetBSD-Win2008	-	-	-	-	-	-	-	-	1	0	0	0
FreeBSD-Solaris	0	2	2	4	0	1	0	1	1	1	0	2
FreeBSD-Debian	1	1	2	0	0	0	0	0	0	0	0	0
FreeBSD-Ubuntu	-	-	-	-	0	2	0	0	0	0	0	0
FreeBSD-Red Hat	2	2	2	0	3	2	0	0	1	0	0	0
FreeBSD-Win2000	0	1	0	1	0	1	0	0	1	0	0	-
FreeBSD-Win2003	-	-	-	1	0	1	0	0	1	0	0	0
FreeBSD-Win2008	-	-	-	-	-	-	-	-	1	0	0	0
Solaris-Debian	1	0	0	0	0	0	0	1	0	0	0	0
Solaris-Ubuntu	-	-	-	-	0	1	0	1	0	0	0	0
Solaris-Red Hat	1	0	1	1	0	1	0	1	3	0	2	0
Solaris-Win2000	0	1	0	1	0	1	0	5	0	0	1	-
Solaris-Win2003	-	-	-	0	0	1	0	5	1	0	1	0
Solaris-Win2008	-	-	-	-	-	-	-	-	0	0	1	0
Debian-Ubuntu	-	-	-	-	0	8	0	2	2	2	0	0
Debian-Red Hat	8	11	5	0	2	20	0	4	2	0	0	0
Debian-Win2000	0	0	0	0	0	0	0	0	1	0	0	-
Debian-Win2003	-	-	-	0	0	0	0	0	1	0	0	0
Debian-Win2008	-	-	-	-	-	-	-	-	0	0	0	0
Ubuntu-Red Hat	-	-	-	-	3	16	1	5	2	0	0	0
Ubuntu-Win2000	-	-	-	-	0	0	0	0	1	0	0	-
Ubuntu-Win2003	-	-	-	-	0	0	0	0	1	0	0	0
Ubuntu-Win2008	-	-	-	-	-	-	-	-	0	0	0	0
Red Hat-Win2000	0	0	0	1	0	0	0	0	1	0	0	-
Red Hat-Win2003	-	-	-	0	0	0	0	0	2	0	0	0
Red Hat-Win2008	-	-	-	-	-	-	-	-	0	0	0	0
Win2000-Win2003	-	-	-	10	23	44	29	47	44	19	49	-
Win2000-Win2008	-	-	-	-	-	-	-	1	25	16	38	-
Win2003-Win2008	-	-	-	-	-	-	-	2	30	21	94	135
Mean	1.4	2.2	2.0	1.4	1.2	2.9	1.1	2.3	3.4	1.4	4.2	4.1
Std Dev	2.1	2.9	2.8	2.4	4.0	8.2	4.9	7.6	8.5	4.7	16.5	22.2
Max	8	11	11	10	23	44	29	47	44	21	94	135

vulnerability disclosures, we developed a new metric, called the *Common Vulnerability Indicator (CVI)*. This indicator is calculated for a given year y, based on the vulnerabilities that were shared by operating systems A and B over a period of *tspan* previous years. CVI is built to ensure the following desirable properties:

i. $CVI_y(A,B) = 0$ if A and B have no common vulnerabilities in the *tspan* interval;
ii. $CVI_y(A,B) < CVI_y(C,D)$ if A and B shared less vulnerabilities than C and D in each year of the considered period;
iii. $CVI_y(A,B) < CVI_y(C,D)$ if A and B had N common vulnerabilities in the distant past while C and D had N shared vulnerabilities more recently;

Prepared using speauth.cls
DOI: 10.1002/spe

iv. $CVI_{y_2}(A,B) < CVI_{y_1}(A,B)$, with $y_1 < y_2$, if the number of common vulnerabilities for A and B has decreased over the years.

Therefore, CVI is useful for comparison purposes, allowing the identification of OS pairs that have a smaller number of common flaws, while taking into consideration the instant when vulnerabilities were found. This last point is particularly important because operating systems are constantly evolving, potentially getting more (or less) diverse, and consequently, one should take into consideration the time dimension when selecting OS configurations (e.g., OS pairs that have had less common flaws recently are likely better candidates).

CVI is computed as follows:

- First, a weighting factor α_i is defined for each year $i \in \{y - tspan + 1, ..., y-2, y-1, y\}$,

$$\alpha_i = 1 - \frac{y - i}{tspan} \tag{1}$$

- CVI is then obtained using the number of vulnerabilities $v_i(A,B)$ that appeared in both operating systems A and B for every year i from the start of the time span up to reference year y,

$$CVI_y(A,B) = \sum_{i=y-tspan+1}^{y} \alpha_i \cdot v_i(A,B) \tag{2}$$

Table VII presents CVI values for the years 2009 to 2011. We have excluded from this analysis OSes with vulnerability information missing for more than one year over the period, to avoid using incomplete data in the calculation of the indicators. Therefore, the following operating systems were not considered in Table VII: Windows2003, Windows2008, Ubuntu, and OpenSolaris. The CVI values are computed for a *tspan* of 10 years to reflect a reasonable history. It is possible to see that, with the exception of one system (Solaris with FreeBSD/Red Hat/Windows2000 in the Fat Server), in all remaining cases CVI shows a decreasing trend. Several of the OSes that have evolved over a large period are having less reported vulnerabilities, and this causes a decline in shared vulnerabilities in the recent years. For some of the OS pairs the drop in the CVI value is quite significant, becoming almost one third of the 2009 value (NetBSD with Debian or Red Hat). In the Isolated Thin Server case, there are three pairs with $CVI(A,B) = 0$, which shows that they have shared no vulnerabilities during the past 12 years, and are thus particularly good candidates to include in intrusion-tolerant configurations. These systems are Debian with either OpenBSD or FreeBSD or Solaris.

An example of the usefulness of CVI can be seen, e.g., when comparing OpenBSD-Red Hat with Solaris-Red Hat. According to Table VI, both OS pairs had 10 common vulnerabilities in the 2000–2011 period, suggesting that they both provide the same degree of diversity. However, from the CVI values in Table VII, it is apparent that OpenBSD and Red Hat have become more diverse, in recent years, than Red Hat and Solaris, making it advisable, from a diversity standpoint, to choose the former OS pair over the latter.

5.2. *Building Replicated Systems with Diversity*

This section describes three strategies for choosing diverse sets of operating systems. These strategies utilize the data analyzed earlier as a basis to make decisions, by assuming that the information reported by NVD on vulnerabilities can be correlated to the amount of diversity among OSes. Of course there are some caveats associated with this approach, which we discuss in Section 6.1, but the alternatives can be even harder to put in practice, especially if one wants to consider a large number of operating systems. For example, for closed systems (e.g., Windows) it is very difficult to actually determine the level of sharing of OS components, and therefore, diversity estimations based on the code cannot be performed. Moreover, even if this estimate could be obtained, there is the risk that it does not reflect the number of vulnerabilities that occur

Prepared using speauth.cls
DOI: 10.1002/spe

Table VII. CVI for the years 2009 to 2011, with *tspan* of 10 years.

	Fat Server			Isolated Thin Server		
OS	2009	2010	2011	2009	2010	2011
OpenBSD-NetBSD	17.1	13.8	14.8	8.4	7.1	6.9
OpenBSD-FreeBSD	22.2	18.0	18.1	14.1	11.6	10.3
OpenBSD-Solaris	4.0	3.1	3.3	1.8	1.4	0.9
OpenBSD-Debian	1.7	1.5	1.4	0.0	0.0	0.0
OpenBSD-Red Hat	5.0	4.1	3.2	1.6	1.4	1.1
OpenBSD-Win2000	1.8	1.5	-	1.8	1.5	-
NetBSD-FreeBSD	19.7	18.3	18.8	11.0	9.3	8.6
NetBSD-Solaris	4.9	4.0	4.2	1.5	1.1	0.7
NetBSD-Debian	1.4	0.9	0.5	0.6	0.4	0.2
NetBSD-Red Hat	1.8	1.1	0.5	0.6	0.4	0.2
NetBSD-Win2000	1.6	1.4	-	1.6	1.4	-
FreeBSD-Solaris	6.5	5.4	6.3	2.3	1.7	1.2
FreeBSD-Debian	1.3	0.8	0.5	0.0	0.0	0.0
FreeBSD-Red Hat	5.6	4.4	3.3	1.7	1.4	1.1
FreeBSD-Win2000	2.3	1.9	-	2.3	1.9	-
Solaris-Debian	1.0	0.8	0.6	0.0	0.0	0.0
Solaris-Red Hat	5.3	6.5	5.5	1.0	0.7	0.5
Solaris-Win2000	5.5	5.7	-	1.0	0.7	-
Debian-Red Hat	26.1	20.5	15.7	3.2	2.1	1.4
Debian-Win2000	0.9	0.8	-	0.9	0.8	-
Red Hat-Win2000	1.8	1.6	-	0.9	0.8	-

simultaneously in several operating systems (e.g., two distinct implementations of the same flawed algorithm are vulnerable).

The first strategy, called *Common Vulnerability Count Strategy (CVCst)*, is based on raw data collected over a large interval, and is the simplest approach for selecting OS pairs. It should be used when one wants to treat all vulnerabilities, regardless of the time which were reported, as equally important to make choices. The second strategy, *Common Vulnerability Indicator Strategy (CVIst)*, uses the CVI described in the previous section to select OS pairs taking into account the incidence of common vulnerabilities over the years. It is indicated when one wants to give greater importance to more recent vulnerabilities, because it is a weighted sum. The third strategy, *Inter-reporting Times Strategy (IRTst)*, follows a different approach from the previous ones, focusing not so much on common vulnerabilities directly, but on the frequency in which vulnerabilities appear in the two OSes. If one wants to give more importance to the time interval between successive reports of common vulnerabilities, this is the best strategy. Since this last criterion complements the previous two, one could explore strategies CVCst and CVIst in conjunction with IRTst.

For every strategy, we present example OS sets for the Fat Server and Isolated Thin Server configurations. Fat Server configurations can be pessimistic in the sense that they may account for common vulnerabilities in applications that are not present in the servers, while Isolated Thin Server configurations reflect more accurately the expected setup of dedicated servers in a replicated system. An intrusion-tolerant system usually requires $3f + 1$ replicas to tolerate f intrusions (e.g., [1, 6, 5, 12]). Therefore, we will focus on sets with four OSes to deploy a hypothetical replicated system with four replicas, which allows one fault to be tolerated.

As a cautious note, one should take into account that Ubuntu and Windows2008 were first released in 2004 and 2008 respectively, so the data for these two OSes was collected for a smaller number of years. Windows2000 is presented in the tables because there are published vulnerabilities until 2010, although it has been gradually replaced by Windows2003 and Windows2008 in the organizations. Consequently, we do not use Windows2000 when choosing the OS sets. We have excluded OpenSolaris from the study because there is data available for only a limited period. In each strategy and configuration we present two sets: *setCon* is more conservative, since it does not contain Ubuntu and Windows2008; and *setUpdt* is more up-to-date because it can include Ubuntu and Windows2008. When looking at these two sets, one should keep in mind that setCon is selected

DOI: 10.1002/spe

from a group of OSes for which there is a large amount of NVD data, which contributes for a higher confidence on the result. On the other hand, setUpdt uses OSes with different amounts of NVD data, which can cause small levels of inaccuracy when making comparisons (e.g., in the CVCst, any OS pair featuring Windows2008 has zero common vulnerabilities until 2007). One way to address this would be to only consider vulnerabilities that appear later than 2007 when choosing setUpdt. We opted not to follow this approach because it has the drawback of discarding too much data.

5.2.1. Common Vulnerability Count (CVCst) The results from the previous section give a strong indication that it should be possible to choose groups of OSes with few common vulnerabilities over reasonable intervals of time. However, we would like to understand if the data from the NVD database is effective at suggesting these groups of OSes. To address this point, we divided the data in two subsets: the *history period*, comprising the data for the interval 2000 to 2009, and the *observed period*, from 2010 to 2011. The objective is to employ the history period to pick the sets of OSes to use on the replicated system (as if the choice was made at the beginning of 2010), and then use the data for the observed period to verify if these choices would have been adequate, that is, if they have a small (preferably the smallest) number of common vulnerabilities in this period.

CVCst makes decisions based directly on the empirical data for the number of common vulnerabilities across all OS pairs. These data are displayed in Table VIII for OSes with a Fat Server configuration. Numbers to the right and above the diagonal line represent the history period, while numbers to the left and below the line stand for the observed period. For example, the entry corresponding to OpenBSD-Red Hat to the right of the diagonal line has the number 10, which means that these OSes shared 10 vulnerabilities between 2000 and 2009. The equivalent entry, but to the left of the diagonal line, is 0 because they had no common flaws reported in 2010 and 2011. As expected, OS pairs from the same family had the highest counts of common vulnerabilities. The only case where there were more vulnerabilities in the observed period than the history period is for the Windows2008–Windows2003 pair, which is explained by the recent release date of Windows2008. It is interesting to notice, however, that most pairs had zero common vulnerabilities in the observed period.

Table VIII. History/observed period for CVCst with Fat Servers.

	OpenBSD	NetBSD	FreeBSD	Solaris	Debian	Ubuntu	Red Hat	Win2000	Win2003	Win2008
OpenBSD	-	33	43	9	2	3	10	3	2	1
NetBSD	4	-	36	9	4	0	6	3	2	1
FreeBSD	4	6	-	12	4	2	12	4	3	1
Solaris	1	1	2	-	2	2	8	8	7	0
Debian	0	0	0	0	-	14	52	1	1	0
Ubuntu	0	0	0	0	0	-	27	1	1	0
Red Hat	0	0	0	2	0	0	-	2	2	0
Win2000	0	0	0	1	0	0	0	-	216	42
Win2003	0	0	0	1	0	0	0	49	-	53
Win2008	0	0	0	1	0	0	0	38	229	-

2000-2009 (history period)

2010-2011 (observed period)

The strategy for building sets of OSes is based on a simple cost function. Given any potential OS pair A and B that could be added to the set, one can perform a lookup in Table VIII to determine the pair's number of common vulnerabilities in the history period. This number corresponds to the cost of adding this OS pair to the group. Similarly, when including a third OS C, it is possible to find in the table the entries for A–C and B–C and take their sum as the cost of integrating C in the group. When building a set with n OSes, the total cost is the addition of each individual cost for all combinations of OS pairs. The sets that lead to smaller values of total cost are considered the best choices for deployment in the replicated system. Accordingly, based on the table, the best groups of four OSes are:

DOI: 10.1002/spe

- setCon = {*OpenBSD, Solaris, Debian, Windows2003*}, with a total cost of 23;
- setUpdt = {*NetBSD, Solaris, Ubuntu, Windows2008*}, with a total cost of 12.

One however should keep in mind that sometimes the total cost may be only an approximation of the actual number of shared vulnerabilities among the OSes in the set, as certain vulnerabilities might be counted more than once. This will likely not be a problem, since overcounting vulnerabilities provides a conservative estimate, or an estimate worse than reality. For example, setUpdt only has 11 shared vulnerabilities for a total cost of 12, since one of the vulnerabilities appears in three of the OSes (and is therefore included in two table entries).

Next we can check to what extent our choice of the best group of four OSes that we would pick from the history period (2000–2009), as prescribed by CVC cost calculation, remains consistent with the choice of best group of four OSes from the observed period (2010–2011). We see that both setCon and setUpdt have only two shared vulnerabilities. They are not the best sets in the observed period (2010–2011), since there are groups of OSes with zero common vulnerabilities in the observed period (e.g., by replacing Solaris with Red Hat), though they do exhibit a high level of diversity. A graphical representation of the sets as Venn diagrams is available in Figures 5(a) and 5(b). Below the OS name is the total number of vulnerabilities during the observed period, and the number inside each intersection shows the count of common flaws for the corresponding OSes. For example, in setCon with Fat Servers (Figure 5(a)) there is one vulnerability that appears both on Solaris and OpenBSD and another on Solaris and Windows2003.

Table IX. History/observed period for strategy CVCst with Isolated Thin Servers.

2000-2009	OpenBSD	NetBSD	FreeBSD	Solaris	Debian	Ubuntu	Red Hat	Win2000	Win2003	Win2008
OpenBSD	-	13	26	5	0	0	3	3	3	1
NetBSD	1	-	18	4	2	0	2	3	1	1
FreeBSD	1	1	-	6	0	0	3	4	2	1
Solaris	0	0	0	-	0	0	2	2	1	0
Debian	0	0	0	0	-	2	8	1	1	0
Ubuntu	0	0	0	0	0	-	1	1	1	0
Red Hat	0	0	0	0	0	0	-	1	1	0
Win2000	0	0	0	0	0	0	0	-	81	13
Win2003	0	0	0	0	0	0	0	4	-	14
Win2008	0	0	0	0	0	0	0	3	26	-
2010-2011										

Table IX presents the common vulnerabilities with the Isolated Thin Server configuration data. This configuration represents a class of servers that have a dedicated function and are protected against physical intruders. The same approach can be applied as above: first, we choose the best pairs based on the history period to build a set of four OSes; next, we evaluate the sets by comparing the results with the values for the observed period. The history period provides two candidate sets:

- setCon = {*NetBSD, Solaris, Debian, Windows2003*}, with a total cost of 9;
- setUpdt = {*Solaris, Debian, Ubuntu, Windows2008*}, with a total cost of 2.

In the observed period, setCon and setUpdt have no common vulnerabilities, showing that the strategy would have chosen sufficiently diverse groups of OSes. A graphical representation of the sets is displayed in Figures 5(c) and 5(d).

5.2.2. Common Vulnerability Indicator Strategy (CVIst) This strategy employs the CVI value, defined at the beginning of this section, to make decisions about including/excluding particular OSes. Therefore, besides taking advantage of the available data on total counts of shared vulnerabilities, it also uses the information on how these numbers have evolved through the years.

DOI: 10.1002/spe

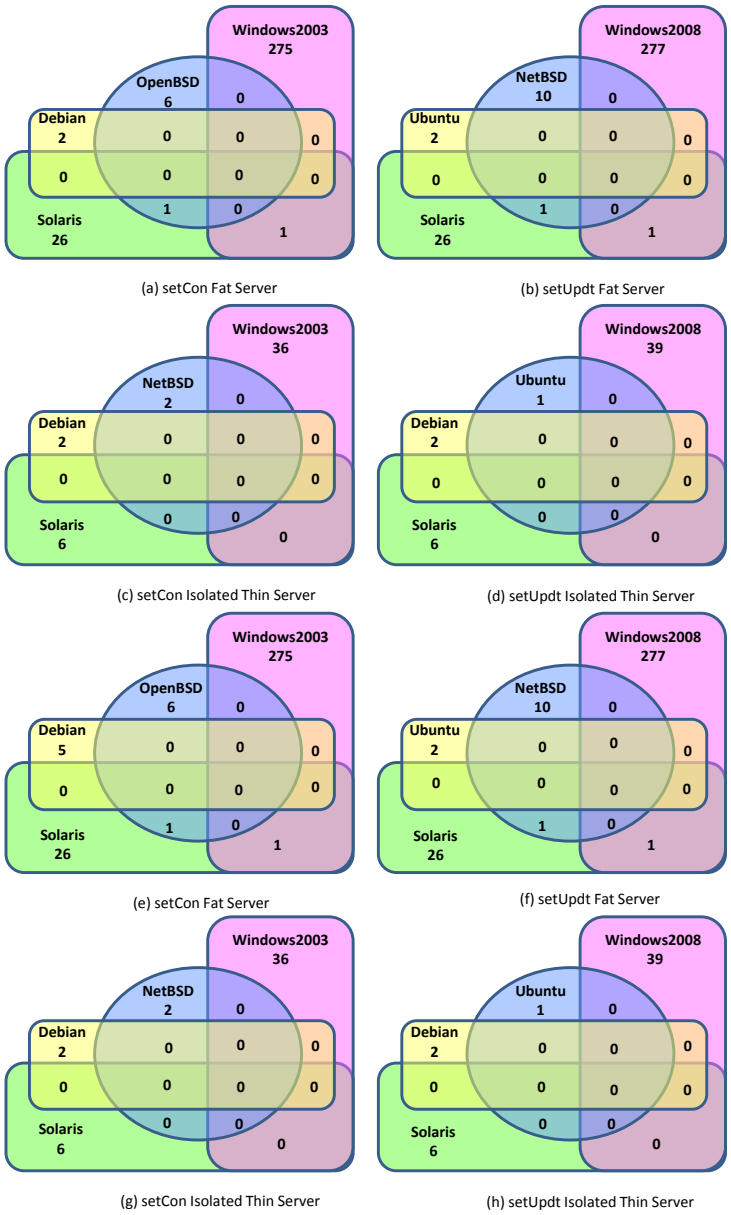

Figure 5. Venn diagrams for vulnerabilities in setCon and setUpdt for strategies CVCst and CVIst in Fat Server and Isolated Thin Server configurations. The first four diagrams (a, b, c and d) represent the results for CVC, and the last four OSes (e, f, g and h) represent the results for CVI.

CVIst is applied by executing the following method, which is based on minimizing a cost function. For a given year and time span, the CVI value is calculated for each of the OS pairs. Typically, one should use the most recent year for which there is available data. The time span should cover a reasonable interval, so that the indicator reflects the trend on discovered vulnerabilities. In some cases, however, one may have to resort to smaller time spans due to lack of data, namely with operating systems released recently. In this case, the indicator will give a higher weight to the vulnerabilities reported in the last year. In CVIst the cost of creating a group with two operating systems A and B is $CVI(A,B)$. Extending this idea to a group of n OSes, the total cost becomes the sum of the individual CVIs for all combinations of OS pairs. In order to choose the best groups, the strategy searches for sets of OSes that together have the smallest total cost.

Prepared using speauth.cls
DOI: 10.1002/spe

Table X. History/observed period for CVIst with Fat Servers.

	OpenBSD	NetBSD	FreeBSD	Solaris	Debian	Ubuntu	Red Hat	Win2000	Win2003	Win2008
OpenBSD	-	17.1	22.2	4.0	1.7	2.5	5.0	1.8	1.5	0.9
NetBSD	4	-	19.7	4.9	1.4	0.0	1.8	1.6	1.5	0.9
FreeBSD	4	6	-	6.5	1.3	1.3	5.6	2.3	2.0	0.9
Solaris	1	1	2	-	1.0	1.5	5.3	5.5	5.6	0.0
Debian	0	0	0	0	-	10.5	26.1	0.9	0.9	0.0
Ubuntu	0	0	0	0	0	-	18.5	0.9	0.9	0.0
Red Hat	0	0	0	2	0	0	-	1.4	1.8	0.0
Win2000	0	0	0	1	0	0	0	-	163.6	39.5
Win2003	0	0	0	1	0	0	0	49	-	0.0
Win2008	0	0	0	1	0	0	0	38	229	-
	2010-2011									

$CVI_{2009}(A,B)$

To evaluate this strategy, we split the time in two intervals as we did for CVCst. Table X presents in the cells for the history period the $CVI_{2009}(A,B)$ for a time span of 10 years in a Fat Server configuration[‡‡]. The cells at left and bottom of the diagonal line correspond to the observed period, and they count as before the number of shared vulnerabilities in 2010 and 2011. After applying CVIst, the following sets are the best with four OSes:

- setCon = {*OpenBSD, Solaris, Debian, Windows2003*}, with a total cost of 14.7;
- setUpdt = {*NetBSD, Solaris, Ubuntu, Windows2008*}, with a total cost of 7.3.

To verify if CVI is a good indicator for selecting diverse sets, we can look at the number of common vulnerabilities in the observed period (2010–2011). By analyzing Table X, it is possible to see that setCon and setUpdt remain good sets, each one with two shared flaws. As before with CVCst, one can find better sets in observed period, where no vulnerabilities appear in common, for example by replacing Solaris with Red Hat. The Venn diagrams for these two sets are shown in Figures 5(e) and 5(f).

Table XI. History/observed period for CVIst with Isolated Thin Servers.

	OpenBSD	NetBSD	FreeBSD	Solaris	Debian	Ubuntu	Red Hat	Win2000	ven Win2003	Win2008
OpenBSD	-	8.4	14.1	1.8	0.0	0.0	1.6	1.8	1.8	0.9
NetBSD	1	-	11.0	1.5	0.6	0.0	0.6	1.6	0.9	0.9
FreeBSD	1	1	-	2.3	0.0	0.0	1.7	2.3	1.5	0.9
Solaris	0	0	0	-	0.0	0.0	1.0	1.0	0.6	0.0
Debian	0	0	0	0	-	1.9	3.2	0.9	0.9	0.0
Ubuntu	0	0	0	0	0	-	0.9	0.9	0.9	0.0
Red Hat	0	0	0	0	0	0	-	0.9	0.9	0.0
Win2000	0	0	0	0	0	0	0	-	58.7	12.5
Win2003	0	0	0	0	0	0	0	4	-	13.5
Win2008	0	0	0	0	0	0	0	3	26	-
	2010-2011									

$CVI_{2009}(A,B)$

Table XI provides the data for applying CVIst in Isolated Thin Server configurations. It is possible to observe that CVI values have significantly decreased when compared to the previous table. The best groups of four OSes in the history period are:

[‡‡]In some cases, we had to use smaller time spans due to the more recent release date of the OSes (e.g., Ubuntu and Windows 2008). When this happened, the CVI value was calculated using the maximum time span that is allowed by the available data.

DOI: 10.1002/spe

- setCon = {*NetBSD, Solaris, Debian, Windows2003*}, with a total cost of 4.5;
- setUpdt = {*Solaris, Debian, Ubuntu, Windows2008*}, with a total cost of 1.9.

By checking the data for the observed period one can see that both sets do not share a single vulnerability, which indicates that the strategy would have made a good selection of OSes (see also the Venn diagrams in Figures 5(g) and 5(h)).

Table XII. Number of two consecutive vulnerabilities occurring in each IRT period, between 2000 and 2011 (with Fat Servers).

	$0 \leq$ IRT ≤ 1	$1 <$IRT ≤ 10	$10 <$IRT ≤ 100	$100 <$IRT ≤ 1000	$1000 <$IRT ≤ 10000
OpenBSD-Win2008	0	0	0	0	0
NetBSD-Ubuntu	0	0	0	0	0
NetBSD-Win2003	0	0	0	0	0
NetBSD-Win2008	0	0	0	0	0
FreeBSD-Win2008	0	0	0	0	0
Solaris-Win2008	0	0	0	0	0
Debian-Win2000	0	0	0	0	0
Debian-Win2003	0	0	0	0	0
Debian-Win2008	0	0	0	0	0
Ubuntu-Win2000	0	0	0	0	0
Ubuntu-Win2003	0	0	0	0	0
Ubuntu-Win2008	0	0	0	0	0
Red Hat-Win2008	0	0	0	0	0
OpenBSD-Win2003	0	0	0	0	1
FreeBSD-Win2003	0	0	0	0	1
Solaris-Debian	0	0	0	0	1
Red Hat-Win2000	0	0	0	0	1
OpenBSD-Win2000	0	0	0	0	2
OpenBSD-Debian	0	0	0	1	0
Solaris-Ubuntu	0	0	0	1	0
NetBSD-Win2000	0	0	0	1	1
FreeBSD-Win2000	0	0	0	2	1
FreeBSD-Ubuntu	0	0	1	0	0
Red Hat-Win2003	0	0	1	0	0
OpenBSD-Solaris	0	0	3	4	2
FreeBSD-Solaris	0	0	5	8	0
FreeBSD-Debian	0	1	0	2	0
OpenBSD-Ubuntu	1	0	0	1	0
NetBSD-Debian	1	0	0	1	0
NetBSD-Solaris	1	0	3	4	1
Solaris-Win2003	1	1	1	4	0
Solaris-Win2000	1	1	1	4	1
NetBSD-Red Hat	2	0	0	3	0
Solaris-Red Hat	2	0	0	7	0
FreeBSD-Red Hat	2	1	2	5	1
Debian-Ubuntu	3	2	3	5	0
OpenBSD-Red Hat	4	0	1	4	0
NetBSD-FreeBSD	4	3	20	14	0
OpenBSD-NetBSD	7	3	14	12	0
Ubuntu-Red Hat	11	2	7	6	0
OpenBSD-FreeBSD	11	5	16	14	0
Debian-Red Hat	22	3	18	8	0
Win2000-Win2008	54	7	15	3	0
Win2000-Win2003	167	29	66	2	0
Win2003-Win2008	222	16	41	2	0

5.2.3. Inter-Reporting Times Strategy (IRTst) This strategy is mainly concerned with the *Inter-Reporting Times (IRT)*, i.e., the number of days between successive reports of common vulnerabilities in different OS pairs, rather than vulnerability counts. The assumption underlying

this strategy is that lower inter-reporting times suggest greater similarity between OSes, and thus it would be advisable, from a diversity standpoint, to select OSes with higher IRTs.

Table XII presents the number of vulnerabilities for each pair of OSes in five IRT intervals, from 0 to 10000 days. The values in the table were obtained in the following manner: first, for a given OS pair A and B, we collected the dates for common vulnerabilities; next, we calculated the IRT in days of every two consecutive vulnerabilities; and then, we counted the number of vulnerabilities that were within each interval. The table is organized such that on the top are the operating systems without common vulnerabilities, which are then followed by the ones that had larger IRT values. Therefore, each horizontal line separates groups of OS pairs that have positive IRT values in the same leftmost column, starting from the rightmost column (i.e., with the longer IRTs). From a diversity perspective, it is interesting to notice that in the table there are 29% of the pairs that do not have two consecutive vulnerabilities, and that 11% only have consecutive vulnerabilities from 1000 days on (last column).

The IRTst strategy allows the selection of OSes with longer IRTs. This criteria is important if one wants to deploy a system that has a short lifetime, e.g., a batching process, which ideally would only be in operation between the discovery of common vulnerabilities. IRTst tries first to select OS pair with zero common vulnerabilities; when this is not possible, it chooses next pair whose common vulnerabilities appear in the rightmost columns. By inspecting the table, we can see that the best two sets of four OSes are:

- setCon = {*OpenBSD, Solaris, Debian, Windows2003*};
- setUpdt = {*NetBSD, Solaris, Ubuntu, Windows2008*}.

Table XIII presents the IRTs for an Isolated Thin Server configuration. Since each OS pair has less common vulnerabilities, this translates often in larger IRTs. The percentage of lines with zero IRTs in all intervals is higher, 51%, but remained the same for the OS pairs that share vulnerabilities with longer IRTs (11%). When applying the strategy to this table, the best sets of OSes are:

- setCon = {*OpenBSD, Solaris, Debian, Windows2003*};
- setUpdt = {*OpenBSD, Debian, Ubuntu, Windows2008*}.

5.2.4. Comparing the three strategies The three strategies explore distinct characteristics of the data to pick the OSes to be deployed. Qualitatively they differ in the method of selection, and potentially the result of applying them to our data could lead to distinct sets being chosen. On the other hand, they all try to find OSes that when placed together in the same system have a low probability of experiencing common vulnerabilities in the future. Therefore, if there is a small collection of OS sets with this property, then all strategies should elect one of these sets as the best choice. This is exactly what we observed with the NVD data, and consequently, the selected best sets are not too different from each other. Only when one needs to find many OS sets with good levels of diversity, such as with the implementation of proactive recovery mechanisms in intrusion-tolerant systems [6, 62], then the distinctions among the strategies start to become apparent. We leave it as future work a more refined study on the comparison of the strategies with larger groups of OS sets.

The OS sets that resulted from the execution of the strategies achieved good levels of diversity when evaluated in the observed period (years 2010 and 2011). As expected from our analysis in Section 4, several of the best sets contained an operating system from each of the families. The exception to this rule occurred with the Linux family, where in a few cases it had two representatives (Debian and Ubuntu) because these OSes had very few vulnerabilities reported in the last three years. Even though the BSD family also had a small number of recent vulnerabilities, since these occasionally affected more OSes, the strategies opted for including just one of the BSD operating systems.

In the next section, we will look into OS versions as a way to increase diversity. For this study, we will use the set that was most selected by the different strategies: {*OpenBSD, Debian, Solaris, Windows2003*} (4 out of 12 choices, considering both Fat Server and Isolated Thin Server configurations).

DOI: 10.1002/spe

Table XIII. Number of two consecutive vulnerabilities occurring in each IRT period, between 2000 and 2011 (Isolated Thin Servers).

	$0 \leq$ IRT ≤ 1	$1 <$IRT ≤ 10	$10 <$IRT ≤ 100	$100 <$IRT ≤ 1000	$1000 <$IRT ≤ 10000
OpenBSD-Debian	0	0	0	0	0
OpenBSD-Ubuntu	0	0	0	0	0
OpenBSD-Win2008	0	0	0	0	0
NetBSD-Ubuntu	0	0	0	0	0
NetBSD-Win2003	0	0	0	0	0
NetBSD-Win2008	0	0	0	0	0
FreeBSD-Debian	0	0	0	0	0
FreeBSD-Ubuntu	0	0	0	0	0
FreeBSD-Win2008	0	0	0	0	0
Solaris-Debian	0	0	0	0	0
Solaris-Ubuntu	0	0	0	0	0
Solaris-Win2003	0	0	0	0	0
Solaris-Win2008	0	0	0	0	0
Debian-Win2000	0	0	0	0	0
Debian-Win2003	0	0	0	0	0
Debian-Win2008	0	0	0	0	0
Ubuntu-Red Hat	0	0	0	0	0
Ubuntu-Win2000	0	0	0	0	0
Ubuntu-Win2003	0	0	0	0	0
Ubuntu-Win2008	0	0	0	0	0
Red Hat-Win2000	0	0	0	0	0
Red Hat-Win2003	0	0	0	0	0
Red Hat-Win2008	0	0	0	0	0
OpenBSD-Win2003	0	0	0	0	1
FreeBSD-Win2003	0	0	0	0	1
Solaris-Red Hat	0	0	0	0	1
Solaris-Win2000	0	0	0	0	1
OpenBSD-Win2000	0	0	0	0	2
Debian-Ubuntu	0	0	0	1	0
NetBSD-Win2000	0	0	0	1	1
FreeBSD-Win2000	0	0	0	2	1
NetBSD-Solaris	0	0	1	2	0
OpenBSD-Solaris	0	0	1	3	0
FreeBSD-Solaris	0	0	1	4	0
NetBSD-Debian	1	0	0	0	0
NetBSD-Red Hat	1	0	0	0	0
Debian-Red Hat	1	0	1	3	2
OpenBSD-Red Hat	2	0	0	0	0
FreeBSD-Red Hat	2	0	0	0	0
OpenBSD-NetBSD	2	0	3	7	1
NetBSD-FreeBSD	2	0	6	9	1
OpenBSD-FreeBSD	6	0	9	10	1
Win2000-Win2008	8	1	3	3	0
Win2003-Win2008	16	2	19	2	0
Win2000-Win2003	35	8	36	5	0

5.3. Exploring Diversity Across OS Releases

If one wants to build systems capable of tolerating a few intrusions, our results show that it is possible to select OSes for the replicas with a small collection of common vulnerabilities. It is hard, however, to support critical services that need to remain correct with higher numbers of compromised replicas or to use some Byzantine fault-tolerant algorithms that trade off performance for extra replicas (e.g., [4, 11, 63]). The number of available operating systems is limited, and consequently, one rapidly runs out of different OSes (e.g., 13 distinct OSes are needed to tolerate $f = 4$ faults in a $3f + 1$ system). On the other hand, our experiments are relatively pessimistic in the sense that they are based on long periods of time and no distinctions are made between OS releases.

DOI: 10.1002/spe

Newer releases of an OS can contain important code changes, and therefore, old vulnerabilities may disappear and/or new vulnerabilities may be introduced. As a result, if we consider (OS, release) pairs, one may augment the number of different systems that do not share vulnerabilities. Nevertheless, one should keep in mind that the use of older OS releases does not come without a cost, namely there might be incompatibilities with the current hardware and some older software packages might be difficult to find.

In the next two sections we explore *n-diverse* sets, where we extend the OS, as an element in the set, to the OS release. First we study a *4-diverse* set, and then a *2-diverse* set built with only two operating systems. We will concentrate on Isolated Thin Server configurations because they correspond to the most common way to deploy intrusion-tolerant systems.

5.3.1. 4-diverse sets Here we analyze in more detail the vulnerabilities for the set {*OpenBSD, Debian, Solaris, Windows2003*} across their releases between 2000 and 2011. Despite of the year of the release, since some vulnerabilities can be inherited from older versions of the code, we will include all vulnerabilities no matter the published date (i.e., even the flaws prior to 2000).

From all releases available for our 4-version replicated system*, we looked at the major releases that had non-zero vulnerabilities. Since OpenBSD follows a fixed six-month release cycle, in this case, we selected one version every three years, which is reasonable given our 12-year time span (taking all 18 releases into consideration would require 154 entries in the table). Therefore, the OS releases that are taken into the study are: OpenBSD 5.0, OpenBSD 4.4, OpenBSD 3.8, OpenBSD 3.2, Solaris 8.0, Solaris 9.0, Solaris 10.0, Solaris 11.0, Debian 2.2, Debian 3.0, Debian 4.0, Debian 5.0, Debian 6.0 and Windows2003.

Table XIV. Common vulnerabilities between OS releases.

OS Versions	Total
Solaris 8.0-Solaris 9.0	21
Solaris 9.0-Solaris 10.0	8
Solaris 8.0-Solaris 10.0	7
OpenBSD 3.2-OpenBSD 3.8	4
OpenBSD 3.2-Solaris 9.0	2
OpenBSD 3.2-Windows2003	2
OpenBSD 3.2-Solaris 8.0	1
OpenBSD 3.8-Windows2003	1
Solaris 8.0-Windows2003	1
Solaris 9.0-Windows2003	1
Solaris 10.0-Windows2003	1
Solaris 10.0-Solaris 11.0	1
Solaris 8.0-Debian 2.2	1
Debian 4.0-Windows2003	1

Table XIV shows the number of common vulnerabilities for each pair of OS-release (pairs with zero values are not displayed). The first observation is that there are many releases within this set of 4 OSes that appear to be free of common flaws. Second, these shared bugs occur more often between releases of the same OS. This is anticipated because more code is re-used within the same operating system. Additionally, one can notice that releases of the same OS typically have less shared vulnerabilities when comparing older and newer versions. This is particularly evident for the OpenBSD and Debian releases. This result is quite promising because it supports our thesis that one should be able to explore diversity across releases, as way to increase the number of available candidates for the construction of the diverse OS sets.

*OpenBSD 2.7, OpenBSD 2.8, OpenBSD 2.9, OpenBSD 3.0, OpenBSD 3.1, OpenBSD 3.2, OpenBSD 3.3, OpenBSD 3.4, OpenBSD 3.5, OpenBSD 3.6, OpenBSD 3.7, OpenBSD 3.8 OpenBSD 3.9, OpenBSD 4.0, OpenBSD 4.1, OpenBSD 4.2, OpenBSD 4.3, OpenBSD 4.4, Solaris 10.0, Solaris 11.0, Solaris 8.0, Solaris 8.1, Solaris 8.2, Solaris 9.0, Solaris 9.1, Debian 2.2, Debian 2.3, Debian 3.0, Debian 3.1, Debian 4.0, Debian 6.0, Debian 6.2, and Windows 2003.

DOI: 10.1002/spe

We can also look with more detail at the vulnerabilities that arise in larger sets of OS releases. In order to understand the impact of these vulnerabilities, we will resort to the classification of vulnerabilities by the CVSS metrics [55]. These metrics indicate how difficult is to exploit a flaw and what is the impact of the attack:

Access Complexity from *Low*: "Specialized access conditions or extenuating circumstances do not exist", to *High*: "Specialized access conditions exist".

Availability Impact from *None*: "There is no impact to the availability of the system" to *Complete*: "There is a total shutdown of the affected resource; The attacker can render the resource completely unavailable".

Confidentiality Impact from *None*: "There is no impact to the confidentiality of the system" to *Complete*: "There is total information disclosure, resulting in all system files being revealed; The attacker is able to read all of the system's data (memory, files, etc.)".

Integrity Impact from *None*: "There is no impact to the integrity of the system" to *Complete*: "There is a total compromise of system integrity; There is a complete loss of system protection, resulting in the entire system being compromised; The attacker is able to modify any files on the target system".

Table XV. Impact of vulnerabilities that affect three or more OS releases.

			Impact on		
CVE	**Affected OSes**	**Access Complexity**	**Availability**	**Confidentiality**	**Integrity**
CVE-2003-0028	OpenBSD 3.2, Solaris 8.0 and 9.0	low	partial	partial	partial
CVE-2003-0466	OpenBSD 2.7 and 3.3, Solaris 9	low	complete	complete	complete
CVE-2004-0790	Solaris 8.0, 9.0 and 10.0	low	partial	none	none
CVE-2004-0791	Solaris 8.0, 9.0 and 10.0	low	partial	none	none
CVE-2004-1307	Solaris 8.0, 9.0 and 10.0	low	partial	partial	partial
CVE-2005-4797	Solaris 8.0, 9.0 and 10.0	low	none	none	partial
CVE-2006-3920	Solaris 8.0, 9.0 and 10.0	low	partial	none	none
CVE-2006-5201	Solaris 8.0, 9.0 and 10.0	high	partial	none	partial
CVE-2007-6180	Solaris 8.0, 9.0 and 10.0	medium	complete	none	complete
CVE-2008-4609	OpenBSD 3.2, 3.8 and Windows2003	medium	partial	partial	partial

Table XV presents the vulnerabilities that affected three or more OS releases with their respective CVSS metrics. It is reasonable to assume that vulnerabilities with low *Access Complexity* and complete *Impact* are the most critical, since they are easy to exploit and they severely affect one or more security attributes. Also relevant are the vulnerabilities with low or medium *Access Complexity* and that have complete *Impact* on at least one security attribute, or partial *Impact* on all attributes. The vulnerabilities in Table XV that meet these criteria can be summarized as follows:

- Three vulnerabilities allow arbitrary code execution (CVE-2003-0028, CVE-2003-0466 and CVE-2004-1307);
- Two vulnerabilities that let denial of service attacks to be performed (CVE-2007-6180 and CVE-2008-4609).

Prepared using speauth.cls
DOI: 10.1002/spe

These results show that over a large period of time, 12 years, there were only a few vulnerabilities that have a significant impact over the various releases under consideration. Moreover, although these security bugs are critical, only three of them affect releases from different families (CVE-2003-0028, CVE-2003-0466 and CVE-2008-4609). This indicates that it is possible to build several good sets with only four operating systems, if the right OS releases are chosen.

5.3.2. 2-diverse sets To reduce the development and maintenance costs of an intrusion tolerant system, it is reasonable to investigate solutions that attempt to decrease the number of distinct OSes but still ensure a high level of security. Here, it makes sense to consider an approach that offers diversity with only two operating systems, while still exploiting the diversity available within the OS releases. In the extreme case, one can select OSes from the same family, for instance, to simplify system management.

To study this sort of solution, we looked at the vulnerabilities that appeared in several versions of Debian and Red Hat (all released after 2000). In total, seven Debian and ten Red Hat releases were considered, which gives a total of 136 combinations. Table XVI provides a summary of the vulnerabilities that were found in NVD for pairs of OS releases (to save space, we omit pairs with zero common vulnerabilities). The reader should notice that Red Hat went through two distinct distributions from 2000, and they can be distinguished by the version number: Red Hat Linux existed between 2000 and 2003, and it has a dot in the version number (e.g., Red Hat 7.1); Red Hat Enterprise Linux is available from 2003, and its releases only have a single digit (e.g., Red Hat 3).

Table XVI. Common vulnerabilities between Debian and Red Hat releases.

OS Versions	Total	**OS Versions**	Total
Debian 2.2 - Debian 2.3	2	Red Hat 7.1 - Red Hat 9.0	2
Debian 3.1 - Debian 4.0	1	Red Hat 7.2 - Red Hat 7.3	5
Debian 4.0 - Red Hat 3	1	Red Hat 7.2 - Red Hat 8.0	5
Debian 4.0 - Red Hat 4	1	Red Hat 7.2 - Red Hat 9.0	2
Debian 4.0 - Red Hat 5	1	Red Hat 7.3 - Red Hat 8.0	5
Red Hat 7.0 - Red Hat 7.1	2	Red Hat 7.3 - Red Hat 9.0	2
Red Hat 7.0 - Red Hat 7.2	1	Red Hat 8.0 - Red Hat 9.0	2
Red Hat 7.1 - Red Hat 7.2	3	Red Hat 3 - Red Hat 4	2
Red Hat 7.1 - Red Hat 7.3	2	Red Hat 3 - Red Hat 5	1
Red Hat 7.1 - Red Hat 8.0	2	Red Hat 4 - Red Hat 5	1

The table only shows 14.7% of the OS releases combinations, which means 85.3% of the OS pairs are free from common flaws. Debian has very few vulnerabilities that are shared between versions. Moreover, accordingly to NVD, these few vulnerabilities only affect two releases, i.e., there is no vulnerability that had an impact over a large period of time. There are two vulnerabilities, CVE-2003-0248 and CVE-2003-0364, that appear in five Red Hat releases (from Red Hat 7.1 to Red Hat 9.0). The first is a vulnerability in the *mxcsr* kernel code which lets attackers modify CPU state registers via a malformed address. This is a critical vulnerability because it has impact on availability, confidentiality and integrity. The second one is in the Red Hat TCP/IP implementation, and allows a DoS by CPU consumption. Three vulnerabilities occur in Red Hat 7.2, 7.3 and 8.0, all related to particular versions and configurations of the OpenSSL cryptographic toolkit. When exploited, they cause a DoS (CVE-2004-0079, CVE-2004-0081 and CVE-2004-0112).

Again, we can observe that there are few high impact vulnerabilities that cross many versions of the same operating system. These results demonstrate that with a careful selection of the Debian and Red Hat releases, it is possible to avoid most of the common vulnerabilities, and apparently it becomes viable to build an intrusion-tolerant replicated system based on only two OSes.

Prepared using speauth.cls
DOI: 10.1002/spe

6. DISCUSSION

6.1. Limitations of data sources and its implications on the study

The numbers we have presented are intriguing and point to a potential for serious security gains from assembling an intrusion-tolerant system using different operating systems. But they are not definitive evidence. Even though the NVD is arguably *the* most complete and referenced database for security vulnerabilities and it is regularly updated with contributions from several sources, there are some uncertainties that remain about the data, which limit the claims we can make about the benefits of diversity to increase security. Ozment [16] points out some problems with the NVD (chronological inconsistency, inclusion, separation of events and documentation); for our purposes, the first two and the last one are the most relevant. "Chronological inconsistency" means that the NVD data has inherent inaccuracies about the dates when vulnerabilities were discovered and when the vulnerable code was released, which not only complicates reasoning about the lifetime of vulnerabilities but also affects the versions that are vulnerable (for instance, sometimes obsolete versions of a product are vulnerable but are not listed in the NVD as such). "Inclusion" refers to the fact that not all vulnerabilities are included in the NVD, only those with a CVE number; as CVE and NVD have gained traction, this has become less of an issue. Finally, there is little documentation about the NVD, and, in the past, the meaning of some fields has occasionally changed without prior notice, which might make comparisons less meaningful. In what follows, we will discuss some other limitations and the implications that they have on the claims we can make about the security benefits of OS diversity:

1. We have looked at the vulnerabilities that are common across the different OSs rather than examining the source code and comparing the similarity / shared code that exists across different OSs.
   Implication: We accept that there may be other ways in which one can check the level of *similarity* or conversely *diversity* between different software systems, one of which is to look at the source code level. We did not take this approach for two main reasons: lack of access to the source code for closed-development systems (such as Microsoft); and also because the level of similarity (or diversity) of the code base may not necessarily tell us how many vulnerabilities the systems have in common. For this reason we chose to study the vulnerability data directly, though studying the code bases (where available) would be a good extension of this study.
2. The NVD does not provide "reproducible scripts" or exploits – probably wisely – which would allow one to check whether the vulnerability can be exploited. From our past experience of working with non-security related bugs [30], a bug report usually contains a script that reproduces the failure that the reporter has observed. Relying solely on the data available in the NVD, it is not possible to confirm that a reported vulnerability is actually exploitable.
   Implication: The lack of exploitability information makes it harder to adequately assess the risk posed by a vulnerability. Caution forces us to consider that all vulnerabilities are exploitable, and must be remediated in due time, a strategy that has obvious implications both in terms of cost and in terms of complexity of management.
3. When a vulnerability is reported for more than one operating system, it is not clear whether the reporter has checked that it has been confirmed to exist in the OSes, or it is just an indication that the vulnerability may exist in each of the operating systems listed.
   Implication: The implications of the previous item apply here as well. Additionally, we have the implication that we cannot claim with certainty whether our estimates of the benefits of diversity, given earlier in the paper, are conservative or optimistic. If a vulnerability has been reported for operating systems A and B but in fact only exists in A, then our estimates are conservative. On the other hand, if the vulnerability has been reported for operating systems A and B only, but in fact it exists additionally in operating systems C and D, then our estimates are optimistic.
4. Although more than 70 organizations (including many important OS vendors) use CVE to identify vulnerabilities, it is not clear if all products are equally represented in the NVD.

DOI: 10.1002/spe

Another related issue is that the vulnerability reporting process is inherently biased, both in timing and in coverage, although not necessarily in an intentional manner. For instance, when a new class of vulnerabilities is disseminated, there is often a surge of new reports involving this class. Finally, not all targets are given the same attention by vulnerability researchers. Software with smaller user bases tend to attract less scrutiny than popular ones, and vulnerabilities with higher impact usually receive more attention. There is even the case where specific vendors are targeted for some reason, as when Oracle claimed their database was "unbreakable" only to have several vulnerabilities disclosed within 24 hours [64], and the rise in exploitation of Adobe software since 2009 [65].
Implication: With any analysis of bug or vulnerability reports from an open database, there is uncertainty about how many of the vulnerabilities are actually reported. This fraction is certainly less than 100%. If all vulnerabilities had the same probability of being reported, the ratio between our predicted vulnerability counts for A and B ($v(A,B)$ – those that affect both products A and B) and A or B ($v(A)$ or $v(B)$ – those that affect only one of the products) would still be the ratio $v(A,B)/v(A)$ or $v(A,B)/v(B)$, respectively. But, in fact, we do not know whether the vulnerabilities of some OS are less likely to be reported in NVD than others (or conversely). It is not clear if the vulnerabilities of some operating systems are reported to the vendors only (or some other vulnerability database) and do not appear in NVD. This again has implications about the claims that we can make about the benefits of diversity, as data entries may be missing which overestimate the benefits of diversity for some products.

5. Our study includes all the vulnerabilities in the NVD whose CPE identifies one of the 11 operating system families that we presented in sections 3 and 4. We also did a significant amount of manual analysis to ensure that ambiguous identifications of OSes could be resolved. However it should be noted that if OS vulnerabilities have been listed in the NVD under different guises (e.g., by specifying as the product name a kernel rather than an OS name), than these types of vulnerabilities are not included in our analysis.
   Implication: We do not know to what extent poor identification of OSes is prevalent in NVD entries. If it is, and if a large number of vulnerabilities for components that are shared by multiple families of OSes (such as kernels) are reported directly under a specific product name rather than an OS name, than our estimates of common vulnerabilities presented in this study would likely be optimistic.
6. To calculate the inter-reporting times in our analysis in section 5.2.3 we have used the report times used in NVD entries. We also cross-checked where possible these dates with another database, namely OSVDB [56]. However, we found some limitations in the OSVDB data. Nearly 70% of the vulnerabilities have incomplete data, e.g., *solution date* is *null*, or is dated as 1970-01-01.
   Implication: The inter-reporting times should be treated with caution, as dates in which vulnerabilities are reported vary depending on the policy and agreements that databases may have with vendors rather than the actual date in which the knowledge of the vulnerability became public.

6.2. *Decisions about deploying diversity*

We have underscored that these results are only *prima facie* evidence for the usefulness of diversity. On average, we would expect our estimates to be conservative as we analyzed aggregated vulnerabilities across releases: common vulnerabilities could be much smaller in a "specific set" of diverse OS releases. But, there are limitations on what can be claimed from the analysis of the NVD data alone without further manual analysis (other than what we have done, e.g., developing/finding and running exploit scripts on every OS for each vulnerability). A better analysis would be obtained if the NVD vulnerability reports were combined with the exploit reports (including exploit counts), and even better if they also had indications about the users' usage profile. However, vendors are often wary of sharing such detailed dependability and security information with their customers. There are partial exploit reports available from other sites (e.g., [57]), but they are incomplete and a

Prepared using speauth.cls
DOI: 10.1002/spe

significant amount of manual analysis is required to match the vulnerabilities with exploits for each operating system.

Given these limitations, *how can individual user organizations decide whether diversity is a suitable option for them, with their specific requirements and usage profiles?* The cost is reasonably easy to assess: costs of the software products, the required middleware (if any), added complexity of management, difficulties with client applications that require vendor-specific features, hardware costs, run-time cost of the synchronization and consistency enforcing mechanisms, and possibly more complex recovery after some failures. The gains in improved security (from some tolerance to 0-day vulnerabilities and easier recovery from some exploits, set against possible extra vulnerabilities due to the increased complexity of the system) are difficult to predict except empirically. This uncertainty will be compounded, for many user organizations, by the lack of trustworthy estimates of their baseline security. We note that, for some users, the evidence we have presented would already indicate that diversity to be a reasonable and relatively cheap precautionary choice, even without highly accurate predictions of its effects. These are users who have serious concerns about security (e.g., high costs for interruptions of service or undetected exploits), and applications which can run on multiple operating systems.

7. CONCLUSIONS

In this paper we presented results from an analysis of 18 years (1994–2011) of vulnerability reports from the NVD database. We analysed 2270 vulnerabilities of eleven operating system distributions. We enriched the dataset with manual fixes and further classification of vulnerabilities, depending on which part of the operating system they affected, into kernel, driver, system software, and applications. The enriched dataset allowed us to perform various analysis to answer the main question that drove our research for this paper: *what are the potential security gains that could be attained from using diverse operating systems in a replicated intrusion-tolerant system?*

The main results could be summarized as follows:

1. In most diverse operating system configurations, significant benefits in security could be observed: a low number of vulnerabilities were found to affect more than one operating system;
2. The number of vulnerabilities that affect more than one operating system depends on how diverse the configuration is: they are higher for OSes from the same family (e.g., BSD) but very low (and in many cases zero) in OSes from different families (e.g., BSD and Windows);
3. The number of common vulnerabilities between OSes can be severely reduced by removing applications (which come pre-installed with an operating system) and by restricting a system to local access. On average 76% of the common vulnerabilities in the study were reduced in this way;
4. We observe few driver vulnerabilities (less than 1%) which is surprising given that previous studies (with non-security-related crash failures) report very high rates of failures in operating systems due to driver errors;
5. We presented several strategies for system designers to choose most diverse operating systems using NVD data depending on whether they: consider all common vulnerabilities as being of equal importance; place greater emphasis on more recent common vulnerabilities (and hence wish to minimise the number of those); are primarily interested in the common vulnerabilities being reported less frequently in calendar time (which would allow them more time to respond to them). Surprisingly, for our dataset all three strategies delivered the same best combination of four operating systems for an intrusion-tolerant configuration (four being the usual number of systems needed to tolerate a Byzantine failure in a replicated system), which is: {*OpenBSD, Debian, Solaris, Windows2003*}.

We discussed in detail that there are limits to the claims we can make on the potential benefits of diversity with operating systems using NVD data alone. However for some users that have severe

DOI: 10.1002/spe

security concerns and applications that are readily portable between operating system platforms the results presented here provide a good case towards employing diversity in their configurations.

There are several strands of further work that are possible:

1. Enrich the NVD dataset with information from exploit databases and patch information from operating system vendors. This would give us far richer information on the vulnerability lifecycle (from reporting to exploitation and patching). However this task requires significant effort to reconcile the differences in the reporting formats of the databases, and the results are likely to remain imperfect due to missing data.
2. Further research on the extent to which we can reason about the benefits of diversity for security from using vulnerability data alone (especially if significant progress on the item above is not forthcoming). This work is currently under way.
3. Analyze the distributions of inter reporting times and enrich them, where possible, with information about operating system usage (say from download rates of these products). This might enable us to reason more accurately about how we can translate these inter-reporting times into vulnerability-free operating time.
4. Examine the diversity that exists between these systems (at least the open source ones) at the source code level. This would then enable analysis of the correlation between the metrics: amount of shared code and number of common vulnerabilities. One would expect the correlation to be strongly positive, but it is difficult to confirm this except with empirical studies.

ACKNOWLEDGEMENT

We would like to thank Paulo Sousa for his early work on this research effort and Peter Bishop for commenting on earlier drafts. This work was partially supported by the EC through project FP7-257475 (MASSIF) and by the FCT through the Multiannual and the CMU-Portugal Programmes, and the project PTDC/EIA-EIA/100894/2008 (DIVERSE). Ilir Gashi has been supported by a Strategic Development Fund (SDF), and a Pump Priming Fund grant, both from City University London.

REFERENCES

1. Veríssimo P, Neves N, Correia M. Intrusion-tolerant architectures: Concepts and design. *Architecting Dependable Systems*, vol. 2677, Lemos R, Gacek C, Romanovsky A (eds.). Springer-Verlag: Berlin, 2003; 3–36.
2. Lamport L, Shostak R, Pease M. The Byzantine generals problem. *ACM Transactions on Programing Languages and Systems* 1982; **4**(3):382–401, doi:10.1145/357172.357176.
3. Avizienis A, Chen L. On the implementation of N-version programming for software fault tolerance during execution. *Proceedings of the IEEE Computer Software and Applications Conference*, Kharagpur, 1977; 173–176.
4. Abd-El-Malek M, Ganger GR, Goodson GR, Reiter MK, Wylie JJ. Fault-scalable Byzantine fault-tolerant services. *Proceedings of the ACM Symposium on Operating Systems Principles*, ACM: New York, 2005; 59–74, doi: 10.1145/1095810.1095817. URL http://doi.acm.org/10.1145/1095810.1095817.
5. Bessani A, Alchieri EP, Correia M, Fraga J. DepSpace: a Byzantine fault-tolerant coordination service. *Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems*, ACM: New York, 2008; 163–176, doi: 10.1145/1352592.1352610. URL http://doi.acm.org/10.1145/1352592.1352610.
6. Castro M, Liskov B. Practical Byzantine fault-tolerance and proactive recovery. *ACM Transactions on Computer Systems* 2002; **20**(4):398–461, doi:10.1145/571637.571640.
7. Castro M, Rodrigues R, Liskov B. BASE: Using abstraction to improve fault tolerance. *ACM Transactions on Computer Systems* 2003; **21**(3):236–269, doi:10.1145/859716.859718. URL http://doi.acm.org/10.1145/859716.859718.
8. Clement A, Kapritsos M, Lee S, Wang Y, Alvisi L, Dahlin M, Riche T. Upright cluster services. *Proceedings of the ACM Symposium on Operating Systems Principles*, ACM: New York, NY, USA, 2009; 277–290, doi: 10.1145/1629575.1629602. URL http://doi.acm.org/10.1145/1629575.1629602.
9. Correia M, Neves N, Veríssimo P. How to tolerate half less one Byzantine nodes in practical distributed systems. *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, IEEE Computer Society: Washington, DC, USA, 2004; 174–183, doi:10.1145/571637.571640.
10. Kapitza R, Behl J, Cachin C, Distler T, Kuhnle S, Mohammadi SV, Schröder-Preikschat W, Stengel K. CheapBFT: Resource-efficient Byzantine fault tolerance. *Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems*, ACM: New York, 2012; 295–308, doi:10.1145/2168836.2168866.
11. Kotla R, Alvisi L, Dahlin M, Clement A, Wong E. Zyzzyva: Speculative Byzantine fault tolerance. *ACM Transactions on Computer Systems* Dec 2009; **27**(4):1–39, doi:10.1145/1323293.1294267.

12. Moniz H, Neves N, Correia M, Verissimo P. RITAS: Services for randomized intrusion tolerance. *IEEE Transactions on Dependable and Secure Computing* 2011; **8**(1):122–136, doi:http://doi.ieeecomputersociety.org/10.1109/TDSC.2008.76.
13. Wood T, Singh R, Venkataramani A, Shenoy P, Cecchet E. ZZ and the art of practical BFT execution. *Proceedings of the ACM Conference on Computer Systems*, ACM: New York, 2011; 123–138, doi:10.1145/1966445.1966457. URL http://doi.acm.org/10.1145/1966445.1966457.
14. Yin J, Martin JP, Venkataramani A, Alvisi L, Dahlin M. Separating agreement from execution for Byzantine fault tolerant services. *Proceedings of the ACM Symposium on Operating Systems Principles*, ACM: New York, 2003; 253–267, doi:10.1145/945445.945470. URL http://doi.acm.org/10.1145/945445.945470.
15. NVD. National Vulnerability Database. http://nvd.nist.gov/ 2011.
16. Ozment A. Vulnerability discovery & software security. PhD Thesis, University of Cambridge 2007.
17. Symantec. Symantec Internet security threat report. http://msisac.cisecurity.org/resources/reports/documents/SymantecInternetSecurityThreatReport2010.pdf April 2011. Accessed: May/2012.
18. Symantec. .
19. Garcia M, Bessani A, Gashi I, Neves N, Obelheiro R. OS diversity for intrusion tolerance: Myth or reality? *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*, IEEE Computer Society: Los Alamitos, 2011; 383–394, doi:http://doi.ieeecomputersociety.org/10.1109/DSN.2011.5958251.
20. Randell B. System structure for software fault tolerance. *SIGPLAN Notices* 1975; **10**(6):437–449, doi:10.1145/390016.808467. URL http://doi.acm.org/10.1145/390016.808467.
21. Knight JC, Leveson NG. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE Trans. Softw. Eng.* Jan 1986; **12**(1):96–109. URL http://dl.acm.org/citation.cfm?id=10677.10688.
22. Hatton L. N-version design versus one good version. *IEEE Softw.* 1997; **14**(6):71–76, doi:10.1109/52.636672. URL http://dx.doi.org/10.1109/52.636672.
23. Joseph MK, Avizienis A. A fault-tolerant approach to computer viruses. *Proceedings of the IEEE Symposium on Research in Security and Privacy*, IEEE Computer Society: Washington, DC, USA, 1988; 52–58.
24. Forrest S, Somayaji A, Ackley DH. Building diverse computer systems. *Proceedings of the Workshop on Hot Topics in Operating Systems*, 1997; 67–72.
25. Hofmeyr SA, Forrest S. ; .
26. Deswarte Y, Kanoun K, Laprie JC. Diversity against accidental and deliberate faults. *Proceedings of the Conference on Computer Security, Dependability, and Assurance: From Needs to Solutions*, IEEE Computer Society: Washington, DC, USA, 1998; 171 –181. URL http://dl.acm.org/citation.cfm?id=519453.793920.
27. Obelheiro R, Bessani A, Lung L, Correia M. How practical are intrusion-tolerant distributed systems? *DI/FCUL TR 06–15*, Department of Informatics, University of Lisbon 2006.
28. Geer D, Bace R, Gutmann P, Metzger P, Pfleeger CP, Quarterman JS, Schneier B. Cyber insecurity: The cost of monopoly. *Technical Report*, Computer & Communications Industry Association 2003. URL http://www.ccianet.org/papers/cyberinsecurity.pdf, accessed: April/2012.
29.
30. Gashi I, Popov P, Strigini L. Fault tolerance via diversity for off-the-shelf products: A study with SQL database servers. *IEEE Transactions on Dependable and Secure Computing* 2007; **4**(4):280–294, doi:10.1109/TDSC.2007.70208.
31. Littlewood B, Popov P, Strigini L. Modeling software design diversity: A review. *ACM Computing Surveys* 2001; **33**(2):177–208, doi:10.1145/384192.384195. URL http://doi.acm.org/10.1145/384192.384195.
32. Littlewood B, Strigini L. Redundancy and diversity in security. *Proceedings of the European Symposium on Research Computer Security*. Springer, 2004; 423–438.
33. Kim J, Malaiya YK, Ray I. Vulnerability discovery in multi-version software systems. *Proceedings of the IEEE High Assurance Systems Engineering Symposium*, IEEE Computer Society: Washington, DC, USA, 2007; 141–148, doi:10.1109/HASE.2007.78. URL http://dx.doi.org/10.1109/HASE.2007.78.
34. Distler T, Kapitza R, Reiser HP. State Transfer for Hypervisor-Based Proactive Recovery of Heterogeneous Replicated Services. *Proceedings of the "Sicherheit, Schutz und Zuverlässigkeit" Conference*, Berlin, 2010; 61–72.
35. Roeder T, Schneider FB. Proactive obfuscation. *ACM Transactions on Computer Systems* 2010; **28**(2):4:1–4:54, doi:10.1145/1813654.1813655. URL http://doi.acm.org/10.1145/1813654.1813655.
36. Miller B, Koski D, Lee C, Maganty V, Murthy R, Natarajan A, Steidl J. Fuzz revisited: A re-examination of the reliability of UNIX utilities and services. *CS-TR 1995–1268*, University. of Wisconsin-Madison 1995.
37. Miller BP, Fredriksen L, So B. An empirical study of the reliability of UNIX utilities. *Communications of the ACM* 1990; **33**(12):32–44, doi:10.1145/96267.96279. URL http://doi.acm.org/10.1145/96267.96279.
38. Koopman P, DeVale J. Comparing the robustness of POSIX operating systems. *Proceedings of the IEEE International Symposium on Fault-Tolerant Computing*, IEEE Computer Society: Washington, DC, USA, 1999; 30–38. URL http://dl.acm.org/citation.cfm?id=795672.796953.
39. Chou A, Yang J, Chelf B, Hallem S, Engler D. An empirical study of operating systems errors. *Proceedings of the ACM Symposium on Operating Systems Principles*, ACM: New York, 2001; 73–88, doi:10.1145/502034.502042. URL http://doi.acm.org/10.1145/502034.502042.
40. Ozment A, Schechter SE. Milk or wine: Does software security improve with age? *Proceedings of the USENIX Security Symposium*, USENIX Association, 2006; 93–104.
41. Anbalagan P, Vouk M. Towards a unifying approach in understanding security problems. *Proceedings of the IEEE International Conference on Software Reliability Engineering*, IEEE Computer Society: Piscataway, 2009; 136–145, doi:10.1109/ISSRE.2009.25. URL http://dx.doi.org/10.1109/ISSRE.2009.25.

42. Alhazmi OH, Malayia YK. Quantitative vulnerability assessment of systems software. *Proceedings of the Annual Reliability and Maintainability Symposium*, 2005; 615–620, doi:doi:10.1109/RAMS.2005.1408432.
43. Anderson RJ. Security in open versus closed systems—the dance of Boltzmann, Coase and Moore. *Conference on Open Source Software: Economics, Law and Policy*, 2002.
44. Rescorla E. Is finding security holes a good idea? *IEEE Security & Privacy* 2005; **3**(1):14–19, doi:10.1109/MSP.2005.17.
45. Alhazmi OH, Malayia YK. Application of vulnerability discovery models to major operating systems. *IEEE Transactions on Reliability* 2008; **57**(1):14 –22, doi:10.1109/TR.2008.916872.
46. Nikora A, Lyu MR. Software reliability measurement experience. *Handbook of software reliability engineering*. McGraw-Hill, Inc.: Hightstown, NJ, USA, 1996; 255–300.
47. Schryen G. Security of open source and closed source software: An empirical comparison of published vulnerabilities. *Proceedings of the Americas Conference on Information Systems*, 2009; 387–387.
48. Gorbenko A, Kharchenko V, Tarasyuk O, Romanovsky A. Using diversity in cloud-based deployment environment to avoid intrusions. *Proceedings of the International Conference on Software Engineering for Resilient Systems*, Springer-Verlag: Berlin, 2011; 145–155. URL http://dl.acm.org/citation.cfm?id=2045537.2045559.
49. CVE. Common Vulnerability Enumeration 2011. Accessed: 2011.
50. Ransbotham S. An empirical analysis of exploitation attempts based on vulnerabilities in open source software. *Proceedings of the Workshop on the Economics of Information Security*, 2010. URL http://weis2010.econinfosec.org/papers/session6/weis2010_ransbotham.pdf, accessed: 2011.
51. Rajnovic D. Monoculture – the other side. BlackHatEuropeBriefings 2011.
52. CVE. CVE terminology. http://cve.mitre.org/about/terminology.html 2011.
53. CPE. Common platform enumeration. http://cpe.mitre.org/ 2011. Accessed: 2011.
54. W3Techs. Usage statistics and market share of Mac OS for websites. http://w3techs.com/technologies/details/os-macos/all/all 2012. Accessed: 2011.
55. CVSS. Common vulnerability scoring system. http://www.first.org/cvss/cvss-guide.html 2011. Accessed: 2011.
56. OSVDB. Open source vulnerability database. http://osvdb.org/ 2011. Accessed: 2011.
57. CVE Details. CVE details website. http://www.cvedetails.com/ 2011.
58. Ganapathi A, Ganapathi V, Patterson D. Windows XP kernel crash analysis. *Proceedings of the Large Installation System Administration Conference*, 2006; 12–22.
59. RedHat. Does CVE-2008-4609 affect Red Hat Enterprise Linux? https://access.redhat.com/kb/docs/DOC-18730 2009. Accessed: 2011.
60. Microsoft. Support for Windows 2000 and Windows XP Service Pack 2 (SP2) ends on July 13, 2010. http://support.microsoft.com/gp/lifean46 2010. Accessed: 2011.
61. Shankland S. Oracle apparently shuts doors on OpenSolaris. http://news.cnet.com/8301-30685_3-20013687-264.html 2010. Accessed: 2011.
62. Sousa P, Bessani A, Correia M, Neves N, Verissimo P. Highly available intrusion-tolerant services with proactive-reactive recovery. *IEEE Transactions on Parallel and Distributed Systems* 2010; **21**:452–465, doi:http://dx.doi.org/10.1109/TPDS.2009.83. URL http://dx.doi.org/10.1109/TPDS.2009.83.
63. Serafini M, Bokor P, Dobre D, Majuntke M, Suri N. Scrooge: Reducing the costs of fast Byzantine replication in presence of unresponsive replicas. *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*, 2010; 353 –362, doi:10.1109/DSN.2010.5544295.
64. Litchfield D. Hackproofing Oracle Application Server. *Whitepaper*, NGS Software Insight 2002.
65. McAfee Labs. 2010 threat predictions. http://www.mcafee.com/us/local_content/white_papers/7985rpt_labs_threat_predict_1209_v2.pdf 2009. Accessed: 2011.