OS diversity for intrusion tolerance: Myth or reality?

CONFERENCE PAPER · JUNE 2011

DOI: 10.1109/DSN.2011.5958251 · Source: DBLP

5 AUTHORS, INCLUDING:

Alysson N Bessani
University of Lisbon
115 PUBLICATIONS **735** CITATIONS
SEE PROFILE

Nuno Ferreira Neves
University of Lisbon
142 PUBLICATIONS **1,662** CITATIONS
SEE PROFILE

Rafael R. Obelheiro
Universidade do Estado de Santa Catarina
25 PUBLICATIONS **133** CITATIONS
SEE PROFILE

OS Diversity for Intrusion Tolerance: Myth or Reality?

Miguel Garcia*, Alysson Bessani*, Ilir Gashi†, Nuno Neves* and Rafael Obelheiro‡
*LaSIGE, University of Lisbon, Faculty of Sciences – Lisbon, Portugal
†Center for Software Reliability, City University London – London, UK
‡Computer Science Department, State University of Santa Catarina – Joinville, Brazil

***Abstract*—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. We have analyzed the vulnerabilities of 11 different OSes over a period of roughly 15 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, our analysis provides a strong indication that building a system with diverse OSes may be a useful technique to improve its intrusion tolerance capabilities.**

***Keywords*-Diversity, Vulnerabilities, NVD, Operating Systems, Intrusion Tolerance.**

I. Introduction

One important application of Byzantine fault-tolerant protocols is to build intrusion-tolerant systems, which are able to keep functioning correctly even if some of their parts are compromised. Such protocols guarantee correct behavior in spite of arbitrary faults provided that a minority (usually less than one third [1]) of components are faulty (for an overview of the area see [2]). To respect this condition, system components need to exhibit failure diversity. However, when security is considered, the possibility of simultaneous attacks against several components cannot be dismissed. If multiple components exhibit the same vulnerabilities, they can be compromised by a single attack, which defeats the whole purpose of building an intrusion-tolerant system in the first place. To reduce the probability of common faults, *diversity* can be employed: each component uses different software to perform the same functions, with the expectation that the differences will reduce the occurrence of common vulnerabilities. This is an orthogonal aspect that affects all works on Byzantine fault-tolerant replication (e.g., [3]–[9]).

Nearly all software systems built today rely on off-the-shelf (OTS) components, such as operating systems and database management systems. This is mostly due to the sheer complexity of such components, coupled with benefits such as the perceived lower costs from their use (some of the components may be open-source and/or freely available), faster deployment and the multitude of available options. Most OTS software, however, have not been designed with security as their top priority, which means that they all have their share of security flaws that can be exploited. At times, supposedly secure systems are compromised not due to vulnerabilities in application software but in a more surreptitious manner, by compromising a critical component in their software infrastructure (e.g., the operating system). On the other hand, given the ready availability of OTS software, leveraging OTS components to implement diversity is less complex and more cost-effective than actually developing variants of software. One of the prime examples is the operating system (OS): realistically, people will resort to an OTS operating system rather than build their own. Given the variety of operating systems available and the critical role played by the OS in any system, diversity at the OS level can be a reasonable way of providing good security against common vulnerabilities at little extra cost.

The focus on common vulnerabilities is an important distinctive of this work. Since intrusion tolerance is usually applied to critical systems, it is safe to assume that maximum care will be exercised in protecting system components, including applying all security patches available. However, even an up-to-date system can be compromised through an undisclosed vulnerability (using a 0-day exploit), since patches usually only appear *after* a vulnerability has been publicized. If such a vulnerability affects several components, there is a window of opportunity for compromising many or all of them at the same time.

The main question we address in this paper is: *What are the gains of applying OS diversity on a replicated intrusion-tolerant system?* To answer this question, we have collected vulnerability data from the NIST National Vulnerability Database (NVD) [10] reported in the period between 1994 and 2010 for 11 operating systems. We focus our study on operating systems for several reasons: they offer a good opportunity for diversity, many intrusions exploit OS vulnerabilities, and 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 collected these data and checked how many vulnerabilities affect more than one operating system. We found this number to be relatively low for most pairs of operating systems. This study was then extended to larger numbers of OSes, with similar conclusions for

selected sets. These results suggest that security gains may be achieved if diverse operating systems are employed in replicated systems.

As a cautious note, we do not claim that these results are definite evidence (although they look quite promising). The main problem is that the available reports concern vulnerabilities and not how many intrusions or exploits occurred for each vulnerability; this makes their use in security evaluation more difficult. Complete intrusion and exploit rates would be much more useful as statistical evidence, but they are not widely available. To most practitioners the only direct security evidence available for these products often are the vulnerability reports. It is the lack of detailed intrusion and exploit data and the lack of known approaches that can utilize existing vulnerability reports of OTS components in security evaluation that has motivated the research detailed in this paper. Our contributions can be summarized as:

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 divided in four families (BSD, Solaris, Linux and Windows) that capture different users preferences;
3) 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.

II. Related Work

Design diversity is a classical mechanism for fault tolerance introduced in the 1970s [11]. N-version programming is a technique for creating diverse software components introduced also in those early years [12]. The main idea behind this mechanism is to use N different implementations of the same component, programmed by N different teams, ideally using distinct languages and methodologies. The objective was to achieve fault tolerance, assuming that designs and implementations developed independently will exhibit failure diversity.

The seminal work on using diversity to improve security is due to Joseph and Avizienis [13]. The paper, however, does not focus so much on diversity but on using diverse components to detect the presence of 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 [14], [15]. Randomized compilation techniques to automatically create diversity in applications were proposed but not developed. Taxonomies of diversity techniques for improving security have been introduced in [16], [17]. However, most of these studies lack empirical or statistical evaluation to support their independence claims.

An experimental study of the benefits of adopting diversity of SQL database servers is presented in [18]. The authors analyzed bug reports for four database servers 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 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 V) prevent us from making the same type of study with NVD data.

Given the criticality of operating systems, there are many papers that study the distribution of bugs and vulnerabilities in OS code. Miller et al. [19], [20] 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. Chou et al. [21] 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 [22] 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 [23] 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.

A comparison of the robustness of 15 different POSIX-based operating systems is presented in [24]. This study was based on fault injection: combinations of valid and invalid parameters were supplied to often-used system calls 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 several degrees of severity). Still, their evidence indicates that, from a reliability standpoint, using different operating systems reduces the number of common failure modes.

Some vulnerability discovery models, which attempt to forecast the amount of vulnerabilities found in software, have been proposed [25]–[27]. Alhazmi and Malayia [28] 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 [29], which measure all defects found in a system (not only those that affect security). This conclusion is disputed in [30], 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 [28] and if there are significant differences between open- and closed-source software [30], while our focus is on assessing the degree of independence between different operating systems. Ozment [31] points out some limitations of the NVD database, which we discuss further in Section V.

Littlewood and colleagues [32] 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 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 [33]. They show that, although diversity does not provide complete failure independence (since design faults are correlated to some extent), it is 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.

III. Methodology

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

Data source: We have analyzed OS vulnerability data from the NVD database [10]. NVD uses the *Common Vulnerability Enumeration (CVE)* definition of vulnerability [34], which is presented 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,[1] 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 2010.[2]

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 an 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 the vulnerability publication; and the security attribute(s) 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 do the aggregation of 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 the list of affected configurations. We have collected vulnerabilities reported for 64 Common Platform Enumerations (CPEs) [35]. 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 OS distributions: *OpenBSD, NetBSD, FreeBSD, OpenSolaris, Solaris, Debian, Ubuntu, RedHat*[3]*, Windows 2000, Windows 2003* and *Windows 2008*. These distributions cover the mostly used *server OS products* of the families: BSD, Solaris, Linux and Windows.

[1] See the complete list on http://cve.mitre.org/compatible/alerts_announcements.html.

[2] 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 September 30th 2010.

[3] *RedHat* comprises the "old" Red Hat Linux (discontinued in 2003) and the more recent Red Hat Enterprise Linux (RHEL).

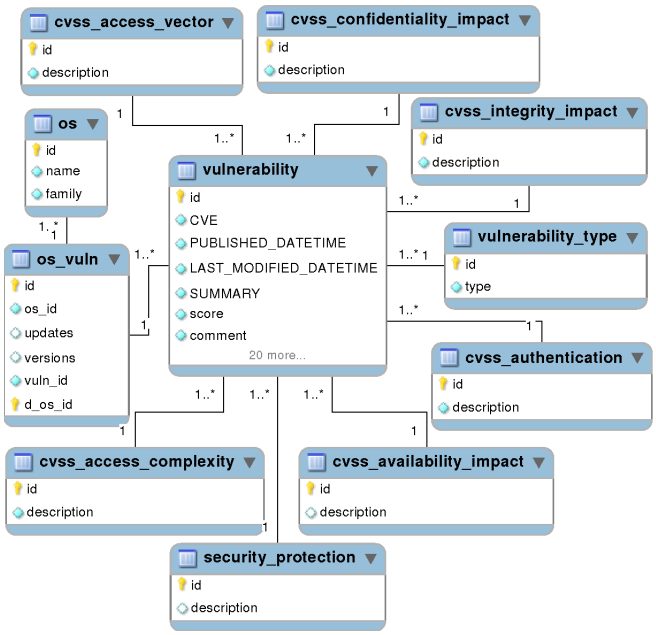


Figure 1. Simplified SQL schema of the database used to store and analyze the NVD data.

The schema of the resulting database is displayed in Figure 1. The tables with prefix *cvss*, *vulnerability_type* and *security_protection* are employed to optimize the database. The most important tables are:

- The *cvss* tables refer directly to the *CVSS* metric of the stored vulnerabilities;
- *vulnerability*: stores some information about a vulnerability (name, publication date, etc.);
- *vulnerability_type*: stores the vulnerability type assigned by us (see Section III-B);
- *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 III-B), 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 example, one of the problems with NVD is that the same product is registered with distinct names for different entries. For example, (”*debian_linux*”,”*debian*”) and (”*linux*”,”*debian*”) are two (product,vendor) pairs we have found for the Debian Linux distribution. This same problem was observed previously by other users of NVD data feeds [36]. Finally, an SQL database is much more convenient to work with than parsing the feeds on demand.

A. Filtering the Data

From the more than 44000 vulnerabilities published by NVD at the time of this study, we selected 2120 vulnerabilities. These vulnerabilities are the ones classified as OS-level vulnerabilities (“/o” on its 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. Due to the uncertainty that surrounds these vulnerabilities, we decided to exclude them from the study. Table I shows the distribution of these vulnerabilities on the analyzed OSes, together with the total number of valid vulnerabilities.

OS	Valid	Unknown	Unspecified	Disputed
OpenBSD	142	1	1	1
NetBSD	126	0	1	2
FreeBSD	258	0	0	2
OpenSolaris	31	0	40	0
Solaris	400	39	109	0
Debian	201	3	1	0
Ubuntu	87	2	1	0
RedHat	369	12	8	1
Win2000	481	7	27	5
Win2003	343	4	30	3
Win2008	118	0	3	0
# distinct vuln.	1887	60	165	8

Table I
DISTRIBUTION OF OS VULNERABILITIES IN NVD.

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 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.

B. Distribution of Vulnerabilities by OS parts

For NVD, an operating system is not only the kernel, but the complete product that is distributed for installation. Therefore an *operating system product* is composed by the kernel, several drivers, optional modules, system software and applications. So, besides knowing how many vulnerabilities affect different operating system products, it is also important to understand what part or module of these systems is compromised by the vulnerability. Since NVD does not provide any information other than the vulnerability

description, we inspected manually each of the 1887 entries and classified them in one of four categories: *Driver, Kernel, System Software* and *Application*. The rationale for this classification is the following:

- **Kernel:** vulnerabilities that affect the TCP/IP stack and other network protocols whose implementation is OS-dependent, file systems, process and task management, core libraries and vulnerabilities derived from processors architectures;
- **Driver:** vulnerabilities that affect drivers for wireless/wired network cards, video/graphic cards, web cams, audio cards, Universal Plug and Play devices, etc;
- **System Software:** vulnerabilities that affect the majority of the software that is necessary to provide common operating system functionalities such as login, shells and basic daemons. We account just 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 products that come with the operating system but that are not needed for basic operations, and in some cases require specific installation: database management systems, messenger clients, text editors and processors, web/email/FTP clients and servers, music/video players, programming languages (compilers and virtual machines), antivirus, Kerberos/LDAP software, games, etc.

The classification above facilitates the analyses of which parts of the operating systems may suffer most from common vulnerabilities, which would influence the architectural decisions of how one designs a diverse system.

IV. OS Diversity Study

This section presents the results of the study. In particular, it presents an overall analysis of the counts of vulnerabilities for each OS component class, and shows how many vulnerabilities affect OS pairs. The section also provides empirical evidence to demonstrate if there are security gains in using diverse OSes when deploying an intrusion-tolerant system.

A. Distribution of OS Vulnerabilities

Vulnerability classification: The descriptions of 1887 vulnerabilities were examined, and then they were assigned to one of the OS component classes presented in the previous section. Table II summarizes the result of this analysis.

The table shows that with the exception of *Drivers*, all OS distributions have a reasonable number of vulnerabilities in each class. In the BSD and Solaris OS families, vulnerabilities appear in higher 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 distributions usually contain a larger set of pre-installed applications, when compared to more stripped down products like BSD family OSes. Therefore, there is a tendency to include more applications in platforms based on these OSes, causing more vulnerabilities of this type to appear in the statistics.

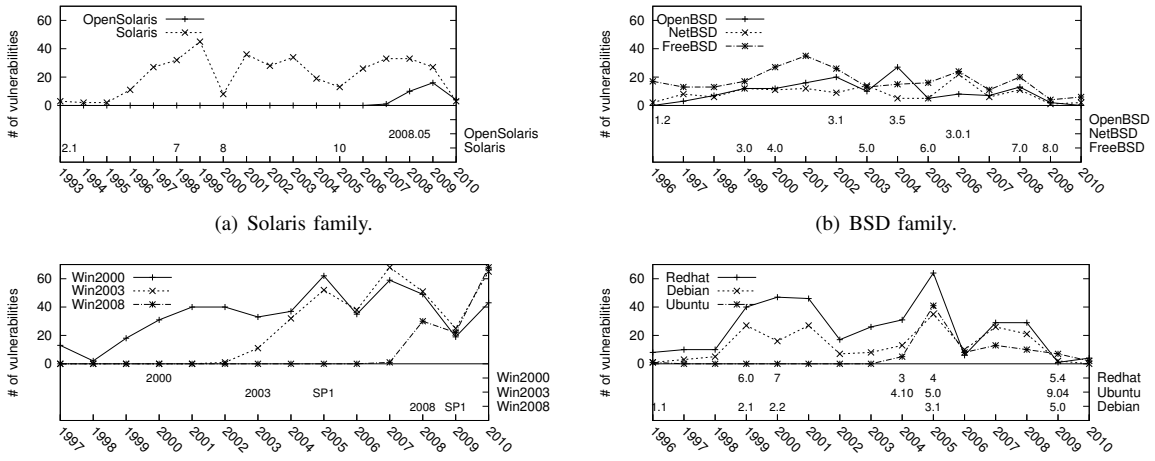
OS	Driver	Kernel	Sys. Soft.	App.	Total
OpenBSD	2	*75*	33	32	142
NetBSD	9	*59*	32	26	126
FreeBSD	4	*147*	54	53	258
OpenSolaris	0	*15*	9	7	31
Solaris	2	*156*	114	128	400
Debian	1	24	34	*142*	201
Ubuntu	2	22	8	*55*	87
RedHat	5	89	93	*182*	369
Windows 2000	3	143	132	*203*	481
Windows 2003	1	95	71	*176*	343
Windows 2008	0	42	14	*62*	118
% Total	1.4%	35.5%	23.2%	39.9%	

Table II
VULNERABILITIES PER OS COMPONENT CLASS.

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 account for a very small percentage of the published OS vulnerabilities. This observation seems to contradict previous studies showing that drivers are the main contributor of crashes [37], and it is somewhat surprising given that drivers usually account for a large percentage of the OS code [21]. One, however, should keep in mind that crash-inducing bugs do not necessarily translate into vulnerabilities, since they might not be exploitable by an adversary (e.g., because the conditions to activate the fault might be extremely hard to force). On the other hand, large and complex codes are a typical breeding ground for programming flaws, and we may experience a rise in driver vulnerabilities in the future.

Temporal distribution of the vulnerabilities: Figure 2 presents the number of vulnerabilities announced per OS for each year, while organizing in separate graphs the OS families. The figure also includes the dates of some of the major releases of the OSes. Certain OSes like Windows 2008 and Ubuntu have several years with zero vulnerabilities because their first distribution is relatively recent.

The graphs lead to some interesting observations. First, it is possible to notice a strong correlation among the peaks and valleys of both the Windows and Linux families, and somewhat to a lesser extent in the BSD family. This could mean that some vulnerabilities might be shared across the family members (see next section for a better discussion). Second, some OS families have less vulnerabilities being reported in the recent past (last 5 years) when compared with the more distant past. This is true both for the BSD and Linux families, which could indicate that the systems are becoming more stable, but also that the employed development process imposes stronger requirements on the quality of the software.



(a) Solaris family.

(b) BSD family.

(c) Windows server family.

(d) Linux family.

Figure 2. Temporal distribution of vulnerability publication data for four operating system families

Finally, it is also important 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 was possible to find Windows 2000 in seven entries earlier than 1999, sharing vulnerabilities with Windows NT. This confirms that Windows 2000 was built with some of the code of Windows NT, but apparently it seems that this code was not fixed from all already known vulnerabilities[4].

B. Common Vulnerabilities

Table III shows the common vulnerabilities that were found in every combination of OS pair over the period of 1994 to (Sept) 2010. The columns with *v(A)* and *v(B)* show the total number of vulnerabilities collected for OS A and B respectively, whereas *v(AB)* is the count of vulnerabilities that affect both the A and B systems. Three filters were applied to the data set: *All* corresponds to all vulnerabilities, representing the raw data; *No Applications* removes from the data set the vulnerabilities classified as *Applications* (see Section III-B); *No Applications and No Local:* same as the previous filter but only considers remotely exploitable vulnerabilities (vulnerabilities with "Network" or "Adjacent Network" values in their CVSS_ACCESS_VECTOR field). The aim of the first filter is to characterize a platform with a reasonable number of installed applications (called a *Fat Server*). The second filter captures only the fundamental OS vulnerabilities, and it basically corresponds to a server platform that, to decrease security risks, is stripped of all applications with the exception of the offered single service (called a *Thin Server*). The third filter represents a similar configuration, but where the machine is physically protected from illegal access and therefore it can only be remotely attacked (called an *Isolated Thin Server*).

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). As expected, OSes from the same family are affected by more common vulnerabilities due to the software components and applications that are reused (e.g., Debian-RedHat or Windows2000-Windows2003). The use of an Isolated Thin Server, when compared with a Fat Server, has a strong impact on the security of the platform because it decreases the number of common vulnerabilities by 56% on average. This means that a significant portion of common vulnerabilities are local (i.e., cannot be exploited remotely) or come from applications that are available on both operating systems.

Table IV shows which part of the OS is 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 common *Kernel* and *System Software* vulnerabilities between Windows 2000 and 2003 indicates that the latter inherits considerable parts of the OS from its predecessor. This same trend is also observed between Windows 2008 and Windows 2003/Windows 2000, 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 family of OS with more common vulnerabilities is BSD, which also re-utilizes several components of the operating system. A somewhat surprising result is the fact that most Linux distributions have much less common vulnerabilities than we anticipated. We inspected manually the vulnerabilities in order to find an explanation, and we discovered that Linux distributions customize both their

[4]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.

Operating Systems	All			No Applications			No App. and No Local		
Pairs (A-B)	$v(A)$	$v(B)$	$v(AB)$	$v(A)$	$v(B)$	$v(AB)$	$v(A)$	$v(B)$	$v(AB)$
OpenBSD-NetBSD	142	126	40	110	100	32	60	41	16
OpenBSD-FreeBSD		258	53		205	48		87	32
OpenBSD-OpenSolaris		31	1		24	1		6	0
OpenBSD-Solaris		400	12		272	10		103	6
OpenBSD-Debian		201	2		59	2		25	0
OpenBSD-Ubuntu		87	3		32	1		10	0
OpenBSD-RedHat		369	10		187	5		58	4
OpenBSD-Windows2000		481	3		278	3		178	3
OpenBSD-Windows2003		343	2		167	2		109	2
OpenBSD-Windows2008		118	1		56	1		26	1
NetBSD-FreeBSD	126	258	49	100	205	39	41	87	24
NetBSD-OpenSolaris		31	0		24	0		6	0
NetBSD-Solaris		400	15		272	12		103	8
Netbsd-Debian		201	3		59	2		25	2
NetBSD-Ubuntu		87	0		32	0		10	0
NetBSD-RedHat		369	7		187	4		58	2
NetBSD-Windows2000		481	3		278	3		178	3
NetBSD-Windows2003		343	1		167	1		109	1
NetBSD-Windows2008		118	1		56	1		26	1
FreeBSD-OpenSolaris	258	31	0	205	24	0	87	6	0
FreeBSD-Solaris		400	21		272	15		103	8
FreeBSD-Debian		201	7		59	4		25	1
FreeBSD-Ubuntu		87	3		32	3		10	0
FreeBSD-RedHat		369	20		187	13		58	5
FreeBSD-Windows2000		481	4		278	4		178	4
FreeBSD-Windows2003		343	2		167	2		109	2
FreeBSD-Windows2008		118	1		56	1		26	1
OpenSolaris-Solaris	31	400	27	24	272	22	6	103	6
OpenSolaris-Debian		201	1		59	1		25	0
OpenSolaris-Ubuntu		87	1		32	1		10	0
OpenSolaris-RedHat		369	1		187	1		58	0
OpenSolaris-Windows2000		481	0		278	0		178	0
OpenSolaris-Windows2003		343	0		167	0		109	0
OpenSolaris-Windows2008		118	0		56	0		26	0
Solaris-Debian	400	201	4	272	59	4	103	25	2
Solaris-Ubuntu		87	2		32	2		10	0
Solaris-RedHat		369	13		187	8		58	4
Solaris-Windows2000		481	9		278	3		178	3
Solaris-Windows2003		343	7		167	1		109	1
Solaris-Windows2008		118	0		56	0		26	0
Debian-Ubuntu	201	87	12	59	32	6	25	10	2
Debian-RedHat		369	61		187	26		58	11
Debian-Windows2000		481	1		278	1		178	1
Debian-Windows2003		343	0		167	0		109	0
Debian-Windows2008		118	0		56	0		26	0
Ubuntu-RedHat	87	369	25	32	187	8	10	58	1
Ubuntu-Windows2000		481	1		278	1		178	1
Ubuntu-Windows2003		343	0		167	0		109	0
Ubuntu-Windows2008		118	0		56	0		26	0
RedHat-Windows2000	369	481	2	187	278	1	58	178	1
RedHat-Windows2003		343	1		167	0		109	0
RedHat-Windows2008		118	0		56	0		26	0
Windows2000-Windows2003	481	343	253	278	167	116	178	109	81
Windows2000-Windows2008		118	70		56	27		26	14
Windows2003-Windows2008	343	118	95	167	56	39	109	26	18

Table III
VULNERABILITIES (1994 TO (SEPT.) 2010): *All* - ALL VULNERABILITIES; *No Application* - NO APPLICATION VULNERABILITIES; *No App. and No Local* - NO APPLICATION VULNERABILITIES AND ONLY REMOTELY-EXPLOITABLE VULNERABILITIES.

kernels 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.

We extended the study of common vulnerabilities to higher numbers of OSes, however, due to space limitations we can only present a summary of the results. When we created combinations of three OSes, we found that there are still 285 vulnerabilities that could compromise the systems (in general these three systems are from the same family). This number is reduced to 102 and 9 vulnerabilities, respectively, in groups of four and five OSes. There are only two vulnerabilities shared by six OSes (with identifiers CVE-2008-1447 and CVE-2007-5365), and one vulnerability that appears in nine OSes (with identifier CVE-2008-4609). The first two cases correspond to protocols that are implemented in a similar manner in various systems, namely DNS and DHCP, and the last one to a well-known denial of service problem in the TCP design.

	OpenBSD	NetBSD	FreeBSD	Solaris	Debian	RedHat	Win2000	Win2003	
OpenBSD	###	9	25	6	0	4	2	1	1994-2005
NetBSD	7	###	15	8	2	2	2	0	
FreeBSD	7	9	###	8	1	5	3	1	
Solaris	0	0	0	###	2	3	3	1	
Debian	0	0	0	0	###	10	0	0	
RedHat	0	0	0	1	1	###	0	0	
Win2000	1	1	1	0	1	1	###	35	
Win2003	1	1	1	0	0	0	46	###	
	2006-2010								

Table V
HISTORY/OBSERVED PERIOD RESULTS FOR ISOLATED THIN SERVERS

OS Pairs	*Driver*	*Kernel*	*Sys. Soft.*	*Total*
Win2000-Win2003	0	40	41	81
OpenBSD-FreeBSD	1	14	17	32
NetBSD-FreeBSD	2	13	9	24
Win2003-Win2008	0	10	8	18
OpenBSD-NetBSD	1	8	7	16
Win2000-Win2008	0	8	6	14
Debian-RedHat	0	5	6	11
FreeBSD-Solaris	0	5	3	8
NetBSD-Solaris	0	4	4	8
OpenBSD-Solaris	0	5	1	6
OpenSolaris-Solaris	0	3	3	6
FreeBSD-RedHat	0	1	4	5
FreeBSD-Win2000	1	3	0	4
OpenBSD-RedHat	0	1	3	4
Solaris-RedHat	0	3	1	4
NetBSD-Win2000	1	2	0	3
OpenBSD-Win2000	0	3	0	3
Solaris-Win2000	0	3	0	3
Solaris-Debian	0	1	1	2
OpenBSD-Win2003	0	2	0	2
FreeBSD-Win2003	0	2	0	2
Debian-Ubuntu	0	0	2	2
NetBSD-Debian	0	0	2	2
NetBSD-RedHat	0	0	2	2
NetBSD-Win2003	0	1	0	1
NetBSD-Win2008	0	1	0	1
OpenBSD-Win2008	0	1	0	1
FreeBSD-Win2008	0	1	0	1
Solaris-Win2003	0	1	0	1
FreeBSD-Debian	0	0	1	1
Debian-Win2000	0	0	1	1
Ubuntu-RedHat	0	0	1	1
Ubuntu-Win2000	0	0	1	1
RedHat-Win2000	0	0	1	1

Table IV
COMMON VULNERABILITIES ON ISOLATED THIN SERVERS.

C. Selecting the OS for the Replicas

Recall that when building an intrusion-tolerant replicated system, one would like to pick a group of OSes for the servers that share a minimum number of vulnerabilities (ideally zero). This selection ensures that the adversary spends more effort and time to break the system, since he or she has to compromise each replica separately.[5] Typical intrusion-tolerant systems require at least $3f+1$ replicas to tolerate f faults (e.g., [3], [7], [9]), and in some specific services they might only need $2f+1$ or more replicas (e.g., [4], [5]).

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 long 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 from the interval 1994 to 2005 (2/3 of the valid vulnerabilities), and the *observed period* for the years between 2006 and 2010 (1/3 of the valid vulnerabilities). The objective is to employ the history period to pick groups of OSes to deploy in an hypothetical intrusion-tolerant system (e.g., BFS [3] or DepSpace [7]), and then use the data on the observed period to verify if the number of shared vulnerabilities is as small as expected. Table V presents the result of the analysis for groups of Isolated Thin Servers. The experiment does not consider Ubuntu, OpenSolaris and Windows 2008 due to lack of meaningful data during the history period. In the table, values above the diagonal line and to the right correspond to common vulnerabilities in pairs of OSes during the history period. Values to the left and below the diagonal line represent the observed period results.

For the *base case* consider that one wants to tolerate a single intrusion, i.e., $f = 1$, in a set of four identical (non-diverse) replicas (e.g., because one wants to keep administrative tasks simple). The best strategy for this scenario would be to pick the OS with the least vulnerabilities during the history period. Debian would be the best choice because it only had 16 vulnerabilities that could be remotely exploited either in the drivers, kernel or system programs. Over the observed period, this system would have 9 shared vulnerabilities (i.e., the ones that were reported for Debian between 2006 and 2010) that could compromise the four replicas of the hypothetical system (see Figure 3).

If one had chosen to employ the "most diverse" operating system group based on what was reported on the history period, then the selected OSes would be Set1 of Figure 3, which is composed by {Windows 2003, Solaris, Debian and OpenBSD}. During the observed period, this set would only have one vulnerability affecting two of the replicas – OpenBSD and Windows 2003. Alternatively, if we had chosen the second "most diverse" configuration, where NetBSD would substitute OpenBSD, then one would add 3 extra common vulnerabilities during the history period. However, during the observed period, one would still only

[5]This is valid under the assumption that the cost to compromise each OS is non-negligible and approximately the same.

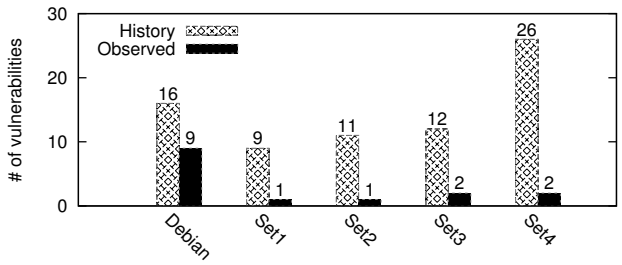


Figure 3. Several configurations of OSes: Debian - only Debian; *Set1* is {Win2003, Solaris, Debian, OpenBSD}; *Set2* is {Win2003, Solaris, Debian, NetBSD}; *Set3* is {Win2003, Solaris, RedHat, NetBSD}; *Set4* is {OpenBSD, NetBSD, Debian, Redhat}.

have a single common vulnerability (between NetBSD and Windows 2003). Therefore, in both configurations of the intrusion-tolerant system, the number of common vulnerabilities would be extremely small, and lower than in the base case.

The results also point out that one can deploy an intrusion-tolerant system with few common vulnerabilities, which is based only on Linux distributions and BSD flavors (Set4 in Figure 3). Since these four OSes can be managed in a relatively similar way, this type of configuration can be extremely useful for organizations that need to operate with tight budgets – e.g., it would not be necessary to hire personnel that knows how to administer Solaris or Windows machines.

Table V shows that it is possible to build a set of six operating systems with few vulnerabilities: two from the BSD family (OpenBSD and NetBSD), one from the Windows family (Windows 2003), the two Linux (Debian and RedHat) and Solaris. By adding one extra operating system, either FreeBSD or OpenSolaris (which only had 6 common vulnerabilities with Solaris in the observed period), we would have seven options available, making it possible to deploy diverse systems with $f = 2$ and $f = 3$, for $3f + 1$ and $2f + 1$ replicas, respectively.

D. Exploring Diversity Across OS Releases

The results from the previous section are encouraging if one wants to build systems capable of tolerating a few intrusions, since 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 performance by extra replicas (e.g., [6], [8]). The number of available operating systems is limited, and consequently, one rapidly runs out of different OSes (e.g., it is necessary 13 distinct OSes to tolerate $f = 4$ in a $3f + 1$ system). On the other hand, our experiments are relatively pessimistic in the sense that they are based in long periods of time and no distinctions are made between OS releases. Newer releases of an OS can contain important code changes, and therefore, current vulnerabilities may not appear in previous versions. As a result, if we consider (OS, release) pairs, one may augment the number of different systems that do not share vulnerabilities.

This section presents some preliminary results about exploring diversity across OS releases. We looked for security advisories (or trackers) available in the various OS websites to determine if they correlate the vulnerabilities patched in each release with the information in NVD. This correlation was found in a meaningful way in four of the OSes under study: NetBSD,[6] Debian,[7] Ubuntu,[8] and RedHat.[9] From all combinations of pairs of these OSes in an Isolated Thin Server configuration, the pair with highest number of common vulnerabilities is Debian-RedHat (see Tables III and IV). Table VI presents the number of common vulnerabilities for three releases of Debian and RedHat, spread along the following years: Debian2.1, 1999; Debian3.0, 2002; Debian4.0, 2007; RedHat6.2*, 2000; RedHat4.0, 2005; RedHat5.0, 2007. One can observe that even though Debian-RedHat shared a total eleven vulnerabilities, the (OS, release) pairs are mostly without common flaws, both in the case of the same OS but distinct releases (left side of the table) and between different operating systems (right side of the table). These same kind of benefits were also reported in a previous work related with non-security bugs for database management systems [18].

OS Versions	Total	**OS Versions**	Total
Debian2.1-Debian3.0	0	Debian3.0-RedHat6.2*	0
Debian2.1-Debian4.0	0	Debian3.0-RedHat4.0	0
Debian3.0-Debian4.0	1	Debian3.0-RedHat5.0	0
RedHat6.2*-RedHat4.0	0	Debian4.0-RedHat6.2*	0
RedHat6.2*-RedHat5.0	0	Debian4.0-RedHat4.0	1
RedHat4.0-RedHat5.0	1	Debian4.0-RedHat5.0	1
		Debian2.1-RedHat6.2*	0
		Debian2.1-RedHat4.0	0
		Debian2.1-RedHat5.0	0

Table VI
COMMON VULNERABILITIES BETWEEN OS RELEASES.

E. Summary of the Results of our Study

The main findings of the study can be summarized as:

1) The number of common vulnerabilities on the studied operating system pairs was reduced by 56% on average if the application and locally-exploitable vulnerabilities are filtered out;
2) More than 50% of the 55 OS pairs studied have at most one non-application, remotely exploitable common vulnerability;
3) The top-3 diverse setups for a four-replica system (tolerating a single failure in typical intrusion-tolerant

[6]http://www.netbsd.org/support/security/release.html
[7]http://security-tracker.debian.org/tracker/
[8]http://people.canonical.com/ ubuntu-security/cve/
[9]https://www.redhat.com/security/data/cve/

system) are: {Windows 2003, Solaris, Debian and OpenBSD}, {Windows 2003, Solaris, Debian and NetBSD} and {Windows 2003, Solaris, RedHat and NetBSD};
4) A preliminary analysis of the diversity among different versions of Debian and RedHat distributions suggests that there are possible setups with the same OS that have a disjoint set of vulnerabilities.
5) There are two vulnerabilities from 2007 and 2008 that affect six OSes, and one vulnerability from 2008 that affected nine OSes;
6) Driver vulnerabilities accounts only for a very small set (less than 1.5%) of all reported OS vulnerabilities.

V. Discussion

A. Limitations of NVD and its Implications on the Study

The numbers we have presented are intriguing and point to a potential for serious security gains from assembling a 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 several uncertainties that remain about the data, which limit the claims we can make about the benefits of diversity to increase security. Ozment [31] 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 benefits of diversity:

1) 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 [18], 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.
2) 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.
3) 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. 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 discovered or disseminated, there is often a surge of new reports involving this class, as it has happened with format string bugs [38] and integer overflows [39]. 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, vulnerabilities with higher impact usually receive more attention, and there is even the case when specific vendors are targeted for some reason, as when Oracle claimed their database was "unbreakable" only to have several vulnerabilities disclosed within 24 hours [40], and the rise in exploitation of Adobe software in the last 15–20 months [41].
   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 AB (mAB – those that affect both products A and B) and A or B (mA or mB – those that affect only one of the products) would still be the ratio mAB/mA or mAB/mB respectively. But, in fact, we do not know whether the vulnerabilities of some

operating systems 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.

B. 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., [36]), but they are incomplete and a 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.

VI. Conclusions

One way to decrease the probability of common vulnerabilities on the replicas of intrusion-tolerant systems is by using diverse OTS software components. In this paper we analyzed the likelihood of common vulnerabilities on an important class of OTS components used in intrusion-tolerant systems: operating systems. We analyzed more than 15 years of vulnerability reports from NVD totaling 2120 vulnerabilities of eleven operating system distributions. The results suggests substantial security gains by using diverse operating systems for intrusion tolerance. We also discussed in detail the limits on the claims we can make about the benefits of diversity from NVD data alone, and discussed what additional data, analysis and clarifications may be needed to increase our confidence about the claims on the benefits of diversity. Despite these limitations, we argue that on average our estimates may be seen as conservative as we analyzed aggregated vulnerabilities across releases – hence common vulnerabilities could be smaller in a "specific set" of diverse OS releases.

VII. Acknowledgments

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 is supported by a Strategic Development Fund (SDF) grant from City University London.

References

[1] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. on Programing Languages and Systems*, vol. 4, no. 3, 1982.

[2] P. Verissimo, N. F. Neves, and M. P. Correia, "Intrusion-tolerant architectures: Concepts and design," in *Architecting Dependable Systems*, ser. LNCS, 2003, vol. 2677.

[3] M. Castro and B. Liskov, "Practical Byzantine fault-tolerance and proactive recovery," *ACM Trans. on Computer Systems*, vol. 20, no. 4, 2002.

[4] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin, "Separating agreement form execution for Byzantine fault tolerant services," in *Proc. of the ACM Symp. on Operating Systems Principles*, 2003.

[5] M. Correia, N. F. Neves, and P. Veríssimo, "How to tolerate half less one Byzantine nodes in practical distributed systems," in *Proc. of the IEEE Symp. on Reliable Distributed Systems*, 2004.

[6] M. Abd-El-Malek, G. Ganger, G. Goodson, M. Reiter, and J. Wylie, "Fault-scalable Byzantine fault-tolerant services," in *Proc. of the ACM Symp. on Operating Systems Principles*, 2005.

[7] A. N. Bessani, E. P. Alchieri, M. Correia, and J. S. Fraga, "DepSpace: a Byzantine fault-tolerant coordination service," in *Proc. of the ACM/EuroSys Conference on Computer Systems*, 2008.

[8] M. Serafini, P. Bokor, D. Dobre, M. Majuntke, and N. Suri, "Scrooge: Reducing the costs of fast Byzantine replication in presence of unresponsive replicas," in *Proc. of the IEEE/IFIP Dependable Systems and Networks*, 2010.

[9] H. Moniz, N. F. Neves, M. Correia, and P. Verissimo, "RITAS: Services for randomized intrusion tolerance," *IEEE Trans. on Dependable and Secure Computing*, vol. 8, no. 1, 2011.

[10] "National Vulnerability Database," http://nvd.nist.gov/.

[11] B. Randell, "System structure for software fault tolerance," *IEEE Trans. on Software Engineering*, vol. 1, no. 2, 1975.

[12] A. Avizienis and L. Chen, "On the implementation of N-version programming for software fault tolerance during execution," in *Proc. of the IEEE Computer Software and Applications Conf.*, 1977.

[13] M. K. Joseph and A. Avizienis, "A fault-tolerant approach to computer viruses," in *Proc. of the IEEE Symp. on Research in Security and Privacy*, 1988.

[14] S. Forrest, A. Somayaji, and D. H. Ackley, "Building diverse computer systems," in *Proc. of the Workshop on Hot Topics in Operating Systems*, 1997.

[15] S. A. Hofmeyr and S. Forrest, "Architecture for an artificial immune system," *Evolutionary Computation*, vol. 8, no. 4, 2000.

[16] Y. Deswarte, K. Kanoun, and J.-C. Laprie, "Diversity against accidental and deliberate faults," in *Computer Security, Dependability, and Assurance: From Needs to Solutions*, 1998.

[17] R. R. Obelheiro, A. N. Bessani, L. C. Lung, and M. Correia, "How practical are intrusion-tolerant distributed systems?" Department of Informatics, University of Lisbon, DI/FCUL TR 06–15, 2006.

[18] I. Gashi, P. Popov, and L. Strigini, "Fault tolerance via diversity for off-the-shelf products: A study with SQL database servers," *IEEE Trans. on Dependable and Secure Computing*, vol. 4, no. 4, 2007.

[19] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of UNIX utilities," *Communications of the ACM*, vol. 33, no. 12, 1990.

[20] B. Miller, D. Koski, C. Lee, V. Maganty, R. Murthy, A. Natarajan, and J. Steidl, "Fuzz revisited: A re-examination of the reliability of UNIX utilities and services," University. of Wisconsin-Madison, CS-TR 1995–1268, 1995.

[21] A. Chou, J.-F. Yang, B. Chelf, S. Hallem, and D. Engler, "An empirical study of operating systems errors," in *Proc. of the ACM Symp. on Operating Systems Principles*, 2001.

[22] A. Ozment and S. E. Schechter, "Milk or wine: Does software security improve with age?" in *Proc. of the USENIX Security Symp.*, 2006.

[23] P. Anbalagan and M. Vouk, "Towards a unifying approach in understanding security problems," in *Proc. of the IEEE Int. Symp. on Software Reliability Engineering*, 2009.

[24] P. Koopman and J. DeVale, "Comparing the robustness of POSIX operating systems," in *Proc. of the IEEE Int. Symp. on Fault-Tolerant Computing*, 1999.

[25] R. J. Anderson, "Security in open versus closed systems—the dance of Boltzmann, Coase and Moore," in *Conf. on Open Source Software: Economics, Law and Policy*, 2002.

[26] E. Rescorla, "Is finding security holes a good idea?" *IEEE Security & Privacy*, vol. 3, no. 1, 2005.

[27] O. H. Alhazmi and Y. K. Malaiya, "Quantitative vulnerability assessment of systems software," in *Proc. of the Annual Reliability and Maintainability Symp.*, 2005.

[28] ——, "Application of vulnerability discovery models to major operating systems," *IEEE Trans. on Reliability*, vol. 57, no. 1, 2008.

[29] M. R. Lyu, Ed., *Handbook of Software Reliability Engineering*. McGraw-Hill, 1995.

[30] G. Schryen, "Security of open source and closed source software: An empirical comparison of published vulnerabilities," in *Proc. of the Americas Conf. on Information Systems*, 2009.

[31] A. Ozment, "Vulnerability discovery & software security," Ph.D. dissertation, University of Cambridge, 2007.

[32] B. Littlewood, P. Popov, and L. Strigini, "Modeling software design diversity: A review," *ACM Computing Surveys*, vol. 33, no. 2, 2001.

[33] B. Littlewood and L. Strigini, "Redundancy and diversity in security," in *Proc. of the European Symp. on Research Computer Security*, 2004.

[34] "CVE terminology," http://cve.mitre.org/about/terminology.html.

[35] "Common platform enumeration," http://cpe.mitre.org/.

[36] "CVE details website," http://www.cvedetails.com/.

[37] A. Ganapathi, V. Ganapathi, and D. Patterson, "Windows XP kernel crash analysis," in *Proc. of the Large Installation System Administration Conference*, 2006.

[38] T. Newsham, "Format string attacks," Guardent, Inc., Tech. Rep., 2000, available from http://www.thenewsh.com/~newsham/format-string-attacks.pdf.

[39] D. Ahmad, "The rising threat of vulnerabilities due to integer errors," *IEEE Security & Privacy*, vol. 1, no. 4, 2003.

[40] D. Litchfield, "Hackproofing Oracle Application Server," NGSSoftware Insight, Whitepaper, 2002.

[41] M. Labs, "2010 threat predictions," Whitepaper, 2009, available from http://www.mcafee.com/us/local_content/white_papers/7985rpt_labs_threat_predict_1209_v2.pdf.