From Analyzing Operating System Vulnerabilities to Designing Multiversion Intrusion-Tolerant Architectures

Anatoliy Gorbenko, Alexander Romanovsky, Olga Tarasyuk, and Oleksandr Biloborodov

***Abstract*—This paper analyzes security problems of modern computer systems caused by vulnerabilities in their operating systems (OSs). Our scrutiny of widely used enterprise OSs focuses on their vulnerabilities by examining the statistical data available on how vulnerabilities in these systems are disclosed and eliminated, and by assessing their criticality. This is done by using statistics from both the National Vulnerabilities Database and the Common Vulnerabilities and Exposures System. The specific technical areas the paper covers are the quantitative assessment of forever-day vulnerabilities, estimation of days-of-grey-risk, the analysis of the vulnerabilities severity and their distributions by attack vector and impact on security properties. In addition, the study aims to explore those vulnerabilities that have been found across a diverse range of OSs. This leads us to analyzing how different intrusion-tolerant architectures deploying the OS diversity impact availability, integrity, and confidentiality.**

***Index Terms*—Days-of-grey-risk, diversity, forever-day vulnerabilities, intrusion tolerance, operating systems (OSs), security, vulnerability, vulnerability databases, vulnerability statistics.**

I. Introduction

IT IS of vital significance for system users and developers alike that information and communication systems are secure. There have been a number of occasions recently, such as those involving the Hollywood Presbyterian Medical Center [1], San Francisco Municipal Transportation Agency [2], or British NHS [3], which have illustrated how exposed modern society is to attacks. The cost of global cyberattacks, such as *Petya* or *WannaCry,* could amount to loss of millions of dollars, harm to our health and survival, and damage to critical infrastructures [4]. It is because our communication equipment, computer systems, and other smart devices suffer from software vulnerabilities that cyberattacks, malware intrusions, and virus infections have been successful.

In general terms, a vulnerability is understood as a weakness that makes it possible for an intruder to damage the information assurance in a system. It has been defined as a software fault that a hacker can employ to access a network or system (MITRE Corporation, [5]). There are various ways in which vulnerabilities can be exploited. Attackers can get commands executed in the normal way, or overcome restrictions in order to gain forbidden access to data, or trigger denial of service and system service termination. The primary source of software vulnerabilities is weaknesses and faults in software design and implementation. Of the 372 updates issued by Microsoft in 2017 for their operating systems (OSs), 228 were security updates for eradicating software vulnerabilities [6]. Of these, 137 were classified as critical.

Both OSs and application software can contain vulnerabilities, yet it is without doubt that security flaws in OSs are that most critical since if they are exploited by attackers, all services and processes executed by the OS can be compromised and illicit access gained to any data that is stored on the exposed machine. Moreover, the threats they pose to system dependability and security are distinct from failures, faults, and errors that have been the traditional focus of the dependability community's efforts.

For instance, in the beginning of May 2017, a global cyber attack using ransomware called Wanna Decryptor (also known as WanaCrypt0r 2.0, WannaCry or WCry) infected more than 300 000 computers in 150 countries, hitting international shipper FedEx, large telecommunications companies in Spain, Portugal, and Argentina, German railway operator Deutsche Bahn, etc. In Britain, the National Health Service (still widely using Windows XP OS in their IT systems) was the worst hit. Many UK hospitals and surgeries were forced to turn away patients and cancel appointments after their IT systems were infected with the ransomware. The attack was initiated through exploiting SMB vulnerability MS17-010 in the Microsoft Windows family of OSs. This paper builds on a number of studies that examine a range of OS security and vulnerability issues [7]–[10]. Our investigation of some novel aspects of security could yield insights that would be significant for not only system

Manuscript received August 13, 2018; revised November 22, 2018; accepted January 28, 2019. Date of publication March 7, 2019; date of current version March 2, 2020. This work was supported by the Engineering and Physical Sciences Research Council /UK STRATA Platform Grant. The work of A. Gorbenko and O. Tarasyuk was supported in part by the TEMPUS ALIOT Grant. Associate Editor: I. Gashi. *(Corresponding author: Anatoliy Gorbenko.)*

A. Gorbenko is with the Leeds Beckett University, LS1 3HE Leeds, U.K. (e-mail: a.gorbenko@leedsbeckett.ac.uk).

A. Romanovsky is with the Newcastle University, NE1 7RU Newcastle-upon-Tyne, U.K. (e-mail: alexander.romanovsky@ncl.ac.uk).

O. Tarasyuk is with the National Aerospace University, Kharkiv 61000, Ukraine (e-mail: o.tarasyuk@csn.khai.edu).

O. Biloborodov is with the Plarium Ukraine LLC, Kharkiv 61072, Ukraine (e-mail: alexander.bright@hotmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2019.2897248

administrators, security engineers, and OS vendors but also ordinary users. It focuses on the following:

1) comparing, by using quantitative analysis and statistics, the vulnerabilities in a number of OSs that have been identified and resolved;
2) assessing the most significant vulnerability metrics including days-of-grey-risk [11], numbers of forever-day [12] vulnerabilities, and their severity for each OS.

This paper expands our earlier work in [13] in a number of ways. First of all, we investigate additional important aspects, such as vulnerability distributions by attack vectors and their impact on different security properties (availability, confidentiality, and integrity); a correlation between vulnerability severity and a vendor's rapidity to fix them; and analyzing which types of vulnerabilities are the most numerous and severe. In addition, we use reported statistics to examine intrusion-tolerant architectures aimed at improving system security using the diversity of OSs and study how diversity can impact surface of attacks targeting different security attributes (availability, integrity, and confidentiality) via common vulnerabilities. Finally, we update our earlier study by adding 2017's vulnerability statistics.

There have been many works, e.g., [14]–[16], studying software diversity as a means for tolerating software faults since the 1970s when the concepts of N-version programming [17] and Recovery Blocks [18] were introduced. Software diversity has been successfully applied in various application domains, including railways, aerospace, and nuclear power station control to improve system reliability.

One of the most challenging parts of the work on applying diversity in practice is the justification of the effectiveness of proposed solutions due to the lack of empirical data. The use of software diversity for security and intrusion tolerance was proposed in earlier studies reported in [19]–[22], which clearly showed the needs for demonstrating the applicability of the proposed architectural solutions and for evaluating their advantages to drive their design.

This paper continues a series of works quantitatively studying common vulnerabilities of intrusion-tolerant systems employing OS diversity, e.g.,[23], [24]. In spite of some similarities between our work and these studies, there are substantial differences. First, Garcia *et al.* do not consider vulnerability statistics in dynamics taking into account a lag between the times when a vulnerability is disclosed and when the OS vendor issues a patch to fix it. Second, in this paper, we analyze additional vulnerability metrics related to different OSs: average days-of-grey-risk, average number of forever-day vulnerabilities, their types and severity. In addition, we examine how OS diversity and common vulnerabilities influence the attack surface and impact various security attributes of the specific intrusion-tolerant architecture. The reported statistics will help system administrators and users to make a justified decision when facing a challenge of choosing the most secure and the least vulnerable OS and their combinations.

The rest of this paper is organized as follows. In Section II, we briefly describe vulnerability databases and studied OSs, discuss the most important vulnerability measures (days-of-grey-risk, forever-day vulnerabilities, and their CVSS severity), present vulnerabilities discovery and patching statistics, and outline the most severe types of vulnerabilities as well as the vulnerabilities discovered in more than one OS. Section III examines diverse intrusion-tolerant architectures and discusses how diversity of OSs affects various security properties: availability, integrity, and confidentiality.

The final part, Section IV, concludes this paper.

II. Background and Research Methodology

The main focus of this paper is to consider dynamical aspects of the vulnerability life cycle. In particular, we study how often new vulnerabilities are discovered, how quick vendors issue patches, fixing vulnerabilities, and how many of yet unfixed vulnerabilities exist in a particular OS at once. With this purpose, our research methodology relies on the following:

1) collecting vulnerability statistics from different datasets and merging them into a single SQL-like database;
2) considering the vulnerability life cycle and disclosure policies that are used by different vulnerability datasets;
3) using the Common Platform Enumerations (CPE, https://cpe.mitre.org/dictionary/) corresponding to the studied OSs to filter vulnerability statistics from the database.

A. Vulnerability Databases and Datasets

There is a wide range of actors that are investing plenty of effort into the discovery and elimination of vulnerabilities, including software vendors, international governmental and nongovernmental organizations, businesses, and individuals. Many of them make their vulnerability datasets publicly available. Among them, the most reputable are as follows:

1) The common vulnerabilities and exposures (CVE) system is a list of established vulnerabilities maintained by MITRE Inc. (cve.mitre.org). Each vulnerability is assigned a unique identifier, CVE-ID, that other vulnerability databases use to synchronize their data with CVE and thus make data exchange between security databases and products possible. Over 18 000 of these identifiers were assigned by MITRE in 2017 alone. The vulnerability description provided in CVE is, however, rather basic and does not include such significant details as a comprehensive list of vulnerable products, vulnerability type, and severity.
2) The National Vulnerability Database (NVD) maintained by the U.S. National Institute of Standards and Technology,[1] builds on and is synchronized with CVE. Unlike CVE, it categorises vulnerabilities by type and severity, provides a specific list of vulnerable software products and additional meta-data following the Common Product Enumeration (CPE) Dictionary, the common weakness enumeration specification (CWE) and the common vulnerability scoring system (CVSS).
3) The vulnerability notes database maintained by CERT.[2]

[1] [Online]. Available: http://web.nvd.nist.gov
[2] [Online]. Available: http://www.kb.cert.org/vuls/

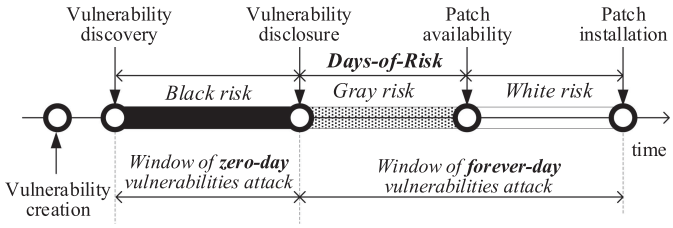

Fig. 1. Vulnerability lifecycle.

4) a vulnerability database offered as a commercial product by the Risk Based Security company[3] can track weaknesses in third-party libraries;
5) securitytracker, a vulnerability dataset available to buy.[4]

Another common way to inform customers about vulnerabilities in software products is vendors publishing security bulletins (e.g., https://technet.microsoft.com/en-us/security/bulletins.aspx). However, the previously widely used open source vulnerability database and Frei's vulnerability database are not accessible any more.

NVD and CVE, the most comprehensive and reliable databases, make vulnerability data available by providing a simple search interface on their websites or daily updated XML data feeds. It would be difficult, however, to directly use their datasets for complex analytics since SQL queries are not supported.

B. Vulnerability Life Cycle and CVE/NVD Disclosure Policies

There have been several studies focusing on the software vulnerability life cycle [11], [25], [26]. In one study [27], its most important milestones were defined in order to put forward its formal model. The common consensus among security analysts and researchers single out the following five major events that make up a typical vulnerability life cycle:

1) a vulnerability is created;
2) it is discovered;
4) it is disclosed;
5) a patch is created;
6) the patch is installed.

The risks of system exposure for time intervals between these events tend to differ. Thus, there is a time of a higher security risk from the moment of vulnerability discovery or disclosure till the moment when a patch is installed to resolve it, referred to as *days-of-risk* [11]. The terms *black, grey,* and *white risk* are used to refer to varying levels of exposure risk and of public awareness of the dangers involved (see Fig. 1). This paper deals with *grey (postdisclosure) risk* associated with the interval between the vulnerability being disclosed and the patch to fix it being provided.

This paper takes the date when a vulnerability is assigned a CVE-ID in CVE as vulnerability *disclosure time*. This is because CVE-IDs are unique identifiers, whereas most other security bulletins and vulnerability databases are seen as secondary since their records are synchronized with them.

[3][Online]. Available: http://www.riskbasedsecurity.com/vulndb
[4][Online]. Available: http://www.securitytracker.com

TABLE I
OSS UNDER INVESTIGATION

Operating system	Release date	Linux kernel version	No of CPE references
Ubuntu Server 12.04	26.04.2012	3.2.x	82
Red Hat Enterprise Linux 6	10.11.2010	2.6.32.x	87
Novell Linux SUSE Enterprise Server 11 SP2	27.02.2012	3.0.13	58
Microsoft Windows Server 2012 R2	18.10.2012	-	12
Apple MacOS Server 10.8	25.06.2012	-	7
Oracle/Sun Solaris 11	09.11.2011	-	9

While it is sometimes possible to derive the time when a patch is produced from vendors' security bulletins, more commonly it is necessary to search vendors' web sites manually in order to extract the relevant information, since typically there are no reporting mechanisms or XML-based data feeds that would allow automatic search and processing.

It has been reported [28] that, for about 75% of vulnerability descriptions, the median time from the moment when they appear in vendor security bulletins till the time when NVD makes them available is seven days. This suggests that National Institute of Standards and Technology (NIST) allows time for a patch to be produced to fix the vulnerability before publishing the detailed information in NVD, implementing what has been called a *responsible disclosure model* [29]. In addition, the median announcement gap varies depending on the vendor: it is two days for Microsoft, five days for Oracle and Apple, ten days for Linux, and 12 days for Novell.

Thus, in our study, we consider a time lag between a vulnerability entries with the same CVE-ID appears in CVE and NVD as a *period-of-grey-risk*. We do not exclude NVD announcement gaps, discussed earlier, during which vulnerability descriptions propagate from vendor's security bulletins to the NVD database (getting ahead, it takes only 6% of the average duration of a *period-of-grey-risk* for studied OSs). This results in a slightly pessimistic estimate of days-of-grey-risk, which, nevertheless, seems to be more secure than their underestimate.

C. OSs Under Study

This study examines the vulnerabilities of six widely used enterprises OSs (see Table I). Our reasons for choosing these particular OSs and their versions include their popularity, the fact that they include both proprietary and open-source types, belong to different families (Windows, Unix/Linux, MacOS), and are sold by different vendors for a range of application domains. This prompted us to consider a series of studies (e.g., [30]–[32]) focusing on the OS market share of web servers, where Linux-based OSs predominate, and of on-premises server, where various versions of Microsoft Windows are most common.

Our aim was to examine vulnerability data over a significant period in order to identify major trends. We also wanted to ensure that our conclusions are based on comprehensive datasets (in NVD and CVE, there is not enough information on the most recent OS versions for statistical analysis). For these reasons, the choice of OS versions was made (see Table I) so as to focus our scrutiny on the six years between late 2011 and late 2017,

analyzing a total of over 2500 vulnerabilities. Even though the OS versions selected have already been replaced by more recent ones, our research demonstrates that new vulnerabilities are still being discovered in the older OS versions. Furthermore, most of these new vulnerabilities can also be found in the latest versions of OSs.

To identify precisely vulnerabilities discovered in a particular OS, we use the CPE Dictionary [33]. The CPE dictionary, maintained by NIST and used by NVD, offers a structured hierarchical naming scheme and a generic syntax for identifying computer systems, software, and packages. Each vulnerability record stored in NVD has a list of CPE references that allows exact identification of all vulnerable products.

Each CPE reference uses the following general syntax: *cpe:/type-of-product{o—OS | a—application software | h—hardware/firmware}:manufacturer:product-name: release:version:subversion(s):platform{x64|x86}*.

In practice, several CPE references can match the same product (e.g., some of CPEs can refer to the whole family of OSs while others can identify the certain OS, version or release). For example, the list of CPE references corresponding to Microsoft Windows Server 2012 R2 consists of 12 entries including the following:

1) cpe:/o:microsoft:windows:::~~~~x64~;
2) cpe:/o:microsoft:windows:::~~~~x86~;
3) cpe:/o:microsoft:all_windows:abstract_cpe;
4) cpe:/o:microsoft:all_windows;
5) cpe:/o:microsoft:windows_server_2012:-, etc.

Lists of CPEs for Ubuntu, Red Hat, and Novell OSs should also be supplemented with CPE entries corresponding to Linux kernels used by each of these OSs. Being a part of an OS, a Linux kernel, nevertheless, is considered by NVD as a separate software product having its own vulnerabilities. For example, the list of CPEs assigned to Linux kernel 3.2.x used by Ubuntu Server 12.04 includes the following:

1) cpe:/o:linux:linux_kernel;
2) cpe:/o:linux:linux_kernel:-;
3) cpe:/o:linux:linux_kernel:3.2;
4) cpe:/o:linux:linux_kernel:3.2::~~~~x86~;
5) cpe:/o:linux:linux_kernel:3.2:rc2;
6) cpe:/o:linux:linux_kernel:3.2.1, etc.

Our study shows that a significant number of vulnerabilities (approximately 40%) disclosed during 2012–2017 in Ubuntu, Red Hat, and Novell are vulnerabilities in their Linux kernels.

D. Research Methodology

Our research methodology is presented in Fig. 2. It consists of the following seven steps:

Step 1: First, we designed and created a MySQL database to aggregate information from the CVE and NVD databases.

Step 2: We developed a software tool that merges together XML data files provided by CVE and NVD, and inserted the joint data set into the MySQL database. The tool consistently updates our MySQL vulnerability database by the following:

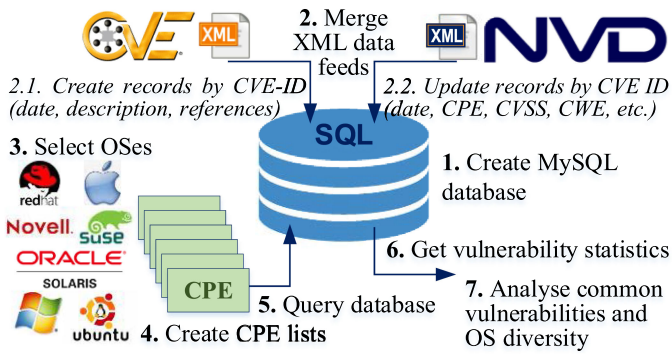

Fig. 2. Research methodology.

1) Downloading XML data feeds from CVE and inserting all new vulnerabilities into the MySQL database, using CVE-ID as a primary key and the CVE date as a vulnerability disclosure time (Step 2.1).
2) Downloading XML data feed from NVD and, if necessary, updating vulnerability records that existed in the MySQL database by CVE-ID (Step 2.2). In particular, if NVD reports a new vulnerability, we set the NVD date as the time when a vulnerability was fixed by a vendor and add CVSS, CWE, and CPE information from NVD in addition to that previously imported from CVE.

Thus, our MySQL database stores both dates associated with the same vulnerability: when a vulnerability is first announced by CVE and when its description appears in NVD. This allows us to estimate the period of *grey risk* as a time lag between them. Because CVE and NVD are updated daily, the tool performs steps 2.1 and 2.2 every day. By now, our MySQL database includes more than 100 000 vulnerability records.

Step 3: At this step, we selected six popular server OSs whose vulnerabilities we wanted to examine.

Step 4: We used the CPE Dictionary to create six lists of CPE references corresponding to each OS. Table I reports how many of the CPE entries have been associated with each OS (the lists themselves can be downloaded from GoogleDrive).[5]

Step 5: The CPE lists created at the previous step were used to query the MySQL database and select a subset of vulnerabilities belonging to certain OSs.

Step 6: At this stage, we run a series of subrequests to collect various vulnerability statistics reported in Section III.

Step 7: At the final step, we studied common OSs vulnerabilities (by analyzing overlaps of the lists of CPE entries assigned to each vulnerability) and investigated how diversity of OSs affects system availability, integrity, and consistency.

The accuracy of the results reported in our work fully depends on the accuracy of the data, reported by CVE and NVD. As mentioned earlier, CVE and NVD are highly reputable vulnerability databases, widely used by many researchers and security analysts that also provide data feeds for third-party security

[5][Online]. Available: https://drive.google.com/open?id=1rToATBng3D4vGL7P7bnoxSywsKe0rDdW

TABLE II
VULNERABILITY DISCLOSURE STATISTICS

Year	Vulnerabilities	Ubuntu	Windows	Red Hat	Novell	MacOS	Solaris
Inherited		15	0	46	26	0	13
2012	Disclosed	64	10	31	32	2	47
	Fixed	32	5	40	36	2	47
	Avg.Sev.	5.21	8.31	5.07	5.13	3.20	4.37
2013	Disclosed	196	59	71	124	60	31
	Fixed	202	51	86	127	59	32
	Avg.Sev.	5.04	7.12	5.09	4.94	4.92	4.72
2014	Disclosed	188	64	72	129	40	37
	Fixed	197	38	58	107	40	35
	Avg.Sev.	5.55	6.71	6.79	5.83	7.13	5.08
2015	Disclosed	251	178	162	52	14	74
	Fixed	223	156	146	80	15	71
	Avg.Sev.	6.06	6.66	5.75	6.67	6.53	5.12
2016	Disclosed	214	204	125	23	48	4
	Fixed	238	156	16	34	48	17
	Avg.Sev.	5.50	5.93	6.87	7.17	6.78	6.60
2017	Disclosed	106	190	53	29	41	5
	Fixed	79	234	59	25	24	9
	Avg.Sev.	6.25	4.50	6.05	6.19	4.04	5.18
Total	Disclosed	1034	705	560	415	205	211
	Fixed	971	640	405	409	188	211
	Avg.Sev.	5.60	6.54	5.94	5.99	5.43	5.18

tools (e.g., vulnerability scanners). Moreover, we assume that being vendor independent MITRE Inc., and NIST, operating CVE and NVD, spend comparatively equal efforts on examining vulnerability of different software products and provide trusted information that can be used as an indicator of software security/quality.

III. OSs' VULNERABILITY STUDY

A. Vulnerability Discovery and Patching Statistics

In this section, we summarize the statistics of vulnerabilities discovered and fixed in different OSs since the 1st of January 2012 and until the 31st of December 2017 (see Table II). In the table, we use the following shorthand for the OSs under investigation:

1) Ubuntu – Ubuntu Server 12.04.
2) Red Hat – Red Hat Enterprise Linux 6.
3) Novell – Novell Linux Enterprise Server 11 SP2.
4) Windows – Microsoft Windows Server 2012 R2.
5) MacOS – Apple MacOS Server 10.8.
6) Solaris – Oracle Solaris 11.

Red Hat Enterprise Linux 6 and Oracle Solaris 11 had been released before the observed period (see Table II). Other OSs (Ubuntu Server 12.04, Novell Linux Enterprise server 11 SP2, Microsoft Windows Server 2012 R2, and Apple Macintosh Server 10.8) were released at the beginning of 2012. It is worth mentioning that on the date of the official release, some of those OSs already had vulnerabilities that earlier had been discovered in previous OS versions. In particular, Ubuntu Server 12.04 inherited 15 of such vulnerabilities, Red Hat Enterprise Linux 6—46, Novell Linux Enterprise server 11 SP2—26, and Oracle Solaris 11—13 vulnerabilities. Such vulnerabilities are reported as " Inherited" in Table II.

TABLE III
AVERAGE DAYS-OF-GREY-RISK STATISTICS

Year	Ubuntu	Windows	Red Hat	Novell	MacOS	Solaris
1999*	-	16	11	-	-	90
2005**	-	24	90	68	55	159
2006**	-	29	107	74	46	168
2012	144	132	243	109	94	89
2013	109	131	119	99	113	81
2014	62	100	108	68	107	69
2015	79	126	101	133	83	58
2016	105	183	130	144	138	210
2017	34	89	80	36	225	49

*taken from [10]; **taken from [7], [9], and [34].

During 2012–2017, the largest number of vulnerabilities (1034) was disclosed in Ubuntu, the smallest (205)—in MacOS. The Red Hat and Novell OSs occupy a middle position having 560 and 415 vulnerabilities, respectively. The greatest number of vulnerabilities in 2007 (190) were discovered in Windows. Cumulative graphs of vulnerabilities disclosed via the CVE and NVD databases during 2012–2017 in the studied OSs are depicted in Fig. 3. It is clear that reporting mechanisms are quite different for different OSs. In particular, curves depicting the numbers of vulnerabilities discovered in Linux-based OSs are less discrete having considerably more "small steps." This means that vulnerabilities are reported quite often by small portions or even individually. At the same time, vulnerabilities in Windows, Solaris, and MacOS are usually batch-reported only a few times per year.

B. Days-of-Grey-Risk Statistics

The number of disclosed vulnerabilities is often used as the main indicator of software insecurity. However, taking into account how fast software vendors react to vulnerabilities discovered in their products is equally important. To compare efforts that different vendors make to solve security issues and to deliver security updates fixing vulnerabilities, we use the *days-of-grey-risk* measure. *Days-of-risk* [11] defines a period of time after a vulnerability is discovered/disclosed and until it is eliminated from a system after a patch installation. It is also known as the "*window of-vulnerability*" or "*days-of-recess*." In this study, we do not take into account possible delays between the times when a vendor issues the patch and when a user or a system administrator actually installs it.

Besides, in many cases, it is impossible to identify when exactly a vulnerability is discovered. In this paper, we investigate so called *grey risk* or *postdisclosure risk* that defines the interval between the vulnerability *disclosure time* and the *date when the patch fixing vulnerability becomes available* [11], [13]. In accordance with our research methodology, discussed in Section II-D, we estimate *days-of-grey-risk* (DoGR) for a particular vulnerability as the period of time between when a vulnerability is initially reported in CVE and when its description appears in NVD.

Table III shows how the average days-of-grey-risk have been changing during 2012–2017 for different OSs. It also includes data reported by other researchers in [7], [9], [10], and [34] for

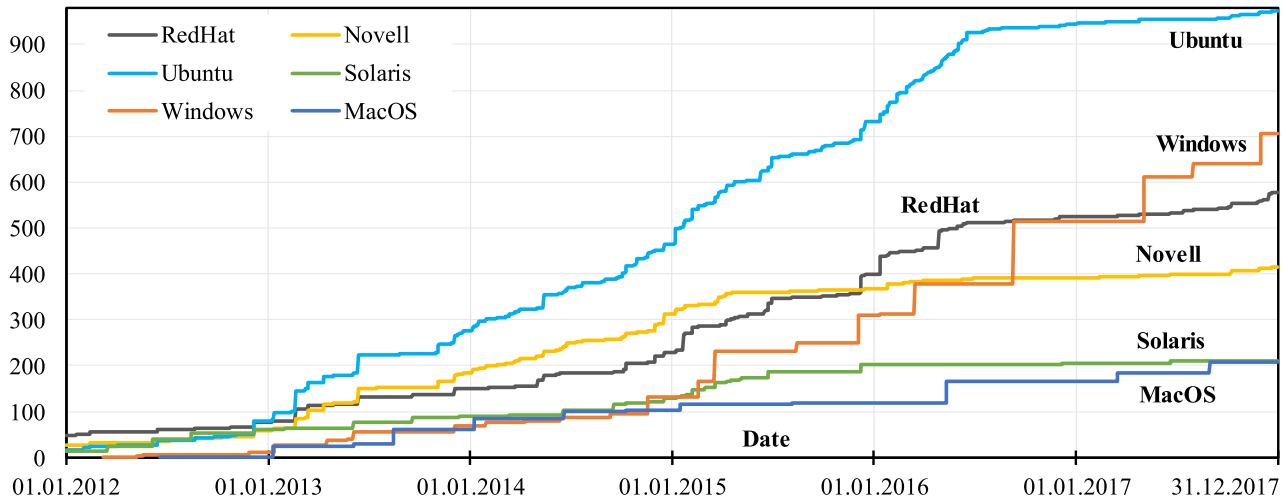

Fig. 3. Cumulative number of disclosed vulnerabilities.

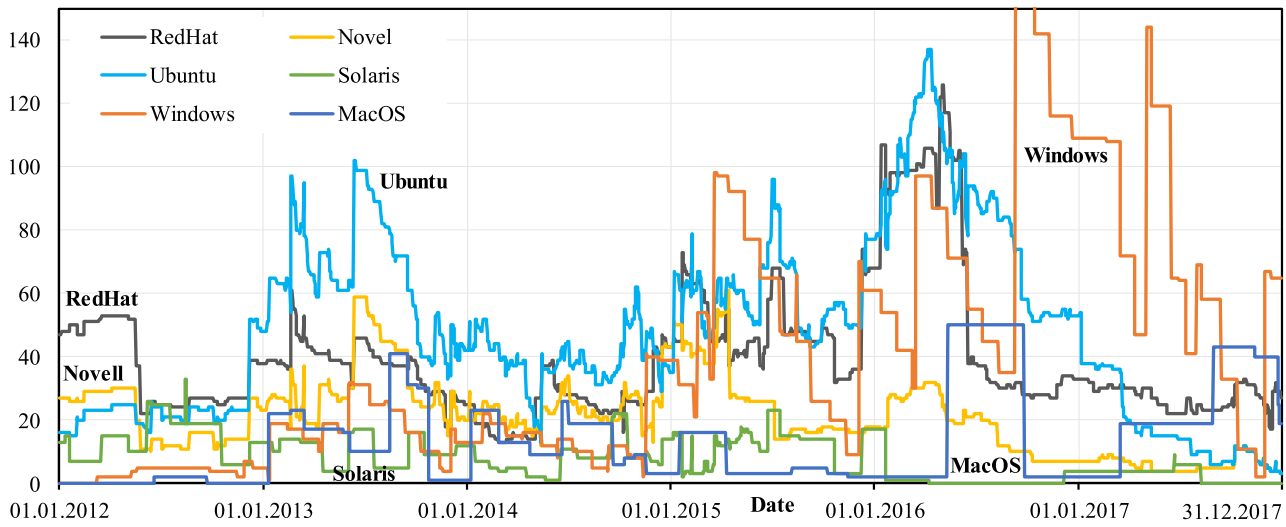

Fig. 4. Forever-day vulnerabilities.

earlier versions of the studied OSs. For instance, according to [10], in 1999, Microsoft spent an average 16 days from vulnerability disclosure to issuing a patch. Red Hat spent only 11 days to fix vulnerabilities while Sun proved itself to be very slow solving security problems in 90 days on average.

In 2006, as reported in [7] and [34], the days-of-grey-risk parameter for Microsoft Windows series of OSs (Windows 2000 Professional and Server, Windows XP, Windows Server 2003) was estimated at 29 on average.

At the same time, it took Red Hat 107 days to deliver security updates for Enterprise Linux 2.1, 3.0, and 4.0, while Sun spent 168 days to do the same for any Solaris version patched in 2006. In addition, it was estimated that Apple Mac OS X and Novell SUSE Linux Enterprise Server and Desktop (versions 8–10) had 46 and 74 days-of-grey-risk, respectively.

Table III shows that since 1999, the period of grey risk has been varying significantly having a form of a sine wave with the general tendency towards increasing the average for all OSs (from 39 days in 1999 to 89 days in 2017). Spikes were observed in 2006, 2012, and 2016. In 2017, the average days-of-grey-risk dropped for all OSs, except MacOS, after having jumped in 2016 when they reached 152 days on average varying between 105 and 210 days. Unfortunately, it still means that after a vulnerability, public disclosure users of the affected OS remain vulnerable and unprotected against potential hacker attacks for months, and the OS vendors are aware of this.

This paper (see Table III) clearly shows that the conclusion by Jeff Jones expressed in a series of his earlier blog posts [7], [11], [35] that Windows is the platform exposing users to risks for the shortest period of time as compared to other OSs is no longer correct in general. However, the diagram on Fig. 7(d) indicates that Microsoft spends considerably less time (22% on average) on fixing the most critical vulnerabilities that CVSS severity scores at 10. At the same time, we can see that since Oracle took ownership of Solaris OS in 2009, the Solaris OS has demonstrated a steady reduction of days-of-grey-risks.

TABLE IV
FOREVER DAY VULNERABILITIES STATISTICS

Year	Forever day vulnerabilities	Ubuntu	Windows	Red Hat	Novell	MacOS	Solaris
2012	Min	15	2	22	10	0	6
	Max	52	7	53	30	2	33
	Average	24	4	36	20	1	14
	Vuln. free days	0	0	0	0	102	0
2013	Min	33	1	18	15	0	4
	Max	102	32	61	59	41	17
	Average	66	18	37	32	17	10
	Vuln. free days	0	0	0	0	9	0
2014	Min	13	2	13	12	1	1
	Max	62	41	47	44	26	22
	Average	39	16	25	23	13	8
	Vuln. free days	0	0	0	0	0	0
2015	Min	35	9	32	14	2	2
	Max	96	98	76	61	16	23
	Average	59	51	49	27	6	12
	Vuln. free days	0	0	0	0	0	0
2016	Min	51	30	27	7	2	0
	Max	137	171	126	32	50	17
	Average	85	83	62	18	20	1
	Vuln. free days	0	0	0	0	0	242
2017	Min	3	2	17	3	2	0
	Max	54	144	33	12	43	9
	Average	17	68	27	6	23	3
	Vuln. free days	0	0	0	0	0	146
Total	Min	3	0	13	3	0	0
	Max	137	171	126	61	50	33
	Average	48	40	40	21	13	8
	Vuln. free days	0	0	0	0	111	388

This led us to conclude that Oracle has been reacting to new vulnerabilities much faster than Sun did.

C. Forever-Day Vulnerability Statistics

Goodin [12] coined a new term "*forever-day vulnerability*" defining a publicly disclosed vulnerability that has not yet been patched and can be hacked any time during system operation. It is in contrast to "*zero-day vulnerabilities*" [27], which are publically undisclosed vulnerabilities that some hackers have already discovered and can exploit.

Using both, the date of vulnerability disclosure and the date when the OS vendor issues a patch to fix it, we can plot graphs of *forever-day vulnerabilities* showing how many of known (already disclosed publicly) but yet unfixed vulnerabilities existed each and every day during 2012–2017 in a particular OS (see Fig. 4). Any OS running with forever-day vulnerabilities is always vulnerable unless the software vendor issues a patch and a system administrator installs it.

Usually, software vulnerabilities are disclosed much faster than vendors manage to fix them. This is why a particular OS can contain up to several dozens of forever-day vulnerabilities at a time. Any of these vulnerabilities could be potentially exploited by hackers to attack the system. Fig. 4 shows that some OSs have only few days (if any) of vulnerability free operation per year. For instance (see Table IV), during 2012–2017, OS Ubuntu, Windows, Red Hat, and Novell did not have known vulnerability free days at all. MacOS had only 111 of such days. It is our hope that OS users and administrators understood and accepted the potential risk of running these systems. In addition, Table IV presents a detailed statistics of forever-day vulnerabilities for each OS during 2012–2017. On an average, Ubuntu OS had 48 of such vulnerabilities every day. OS Windows and Red Hat had 40 forever-day vulnerabilities on average (twice as many as Novell). MacOS and Solaris had the least average number of forever-day vulnerabilities (13 and 8, respectively).

D. Vulnerability Severity and CVSS-Based Statistics

Quantitative evaluation of computer systems vulnerability is a question of great debate with many approaches proposed [36]–[39]. It is clear that the more the vulnerabilities exist in a system, the more that system is prone to hacker attacks.

However, one should also take into account how quick a vendor fixes vulnerabilities, how critical vulnerabilities are, how they impact on security properties, etc.

Vulnerability *severity* is an important characteristic quantifying the impact of vulnerability on system security. NVD has adopted the CVSS to assign severity scores to software vulnerabilities [40]. CVSS comprises of three metric groups, Base, Temporal, and Environmental, each consisting of a set of metrics. The CVSS *Base score* represents the intrinsic and fundamental characteristics of a vulnerability independently of exploits and/or payloads. It is calculated using a group of qualitative metrics taking into account the following:

1) attack vector (local, adjacent network, network);
2) access complexity (high, medium, or low);
3) need for authentication (required or not; multiple or single);
4) vulnerability impact on *confidentiality*, *integrity,* and *availability* (none, partial, or complete); some vulnerabilities impact only one security attribute while others can lead to breaches in two or all three of them.

Temporal and *Environmental* scores are optional. They represent the characteristics of a vulnerability that can change over time (e.g., once the exploit code becomes available) and among user environments (e.g., whether a vulnerable system is exposed publically in the Internet or not). In this section, we consider only CVSS base scores provided by the NVD vulnerability database that are constant over time and user environments. Note that the CVSS vulnerability severity ranges from 0 to 10, with 10 being the most severe.

The average CVSS vulnerability severity scores (Avg.Sev.) for different OSs are presented in Table II. We can see, for example, that vulnerabilities in Oracle Solaris are the least critical, with average severity equal to 5.18. The most severe vulnerabilities have been discovered in Microsoft Windows (the average severity is 6.54) and Novell (the average severity is 5.99).

Fig. 5 shows the percentage of vulnerabilities with different severity levels. Almost a quarter of vulnerabilities discovered in Microsoft Windows, MacOS, and Red Hat are critical (e.g., their CVSS severity scores are in the range [8.0–10.0]). The lowest percentage of critical vulnerabilities (less than 12%) was observed in Solaris.

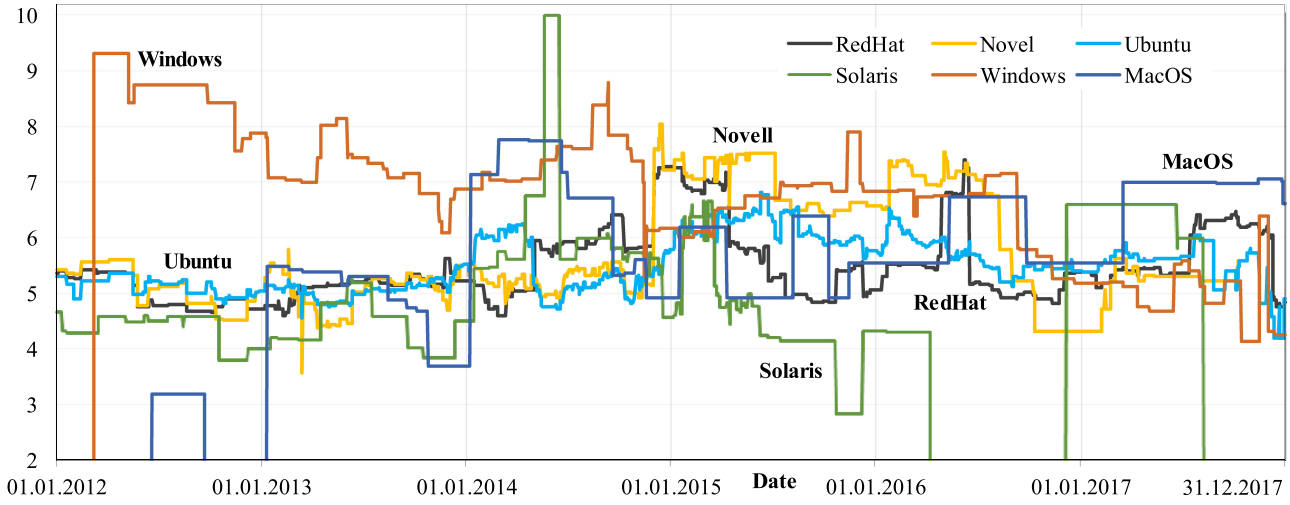

Fig. 5. Average severity of forever-day vulnerabilities.

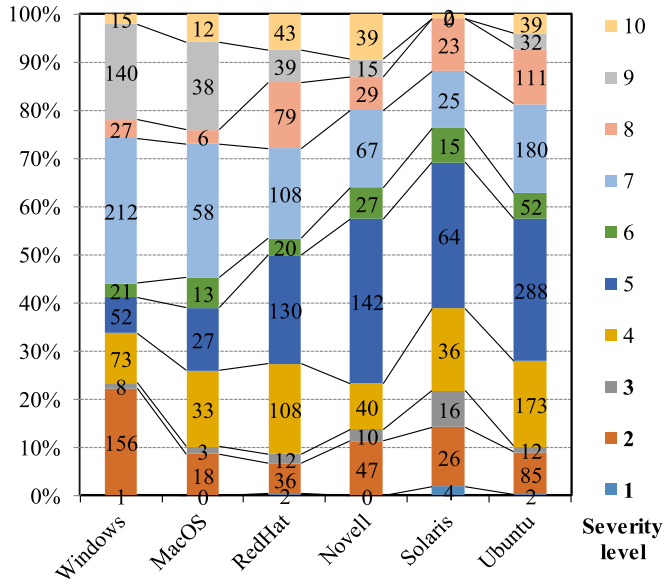

Fig. 6. Vulnerabilities distribution by CVSS severity scores.

It is worth mentioning that *system vulnerability* is a dynamically changing characteristic. It changes every time a new vulnerability is discovered in a system or when a patch fixing one of the previously discovered vulnerabilities is issued by a vendor and applied by a system administrator. Thus, system vulnerability at a particular moment of time can be estimated as a product of the current number of *forever-day-vulnerabilities* (see Fig. 4) and their *average severity*. As shown in Fig. 6, the severity of vulnerabilities disclosed in the Microsoft OS, having the highest value on average, nevertheless, tends to gradually decrease in time. In contrast, the severity of vulnerabilities in Linux- and Unix-based systems is gradually increasing. It is also worth noting that there is no strong correlation between the numbers of forever-day vulnerabilities observed in particular OSs and their average severity.

Table V demonstrates vulnerabilities' distribution against different CVSS criteria: attack vector, need for authentication, and impact on security properties.

It shows, for example, that 75% of vulnerabilities in the Red Hat OS are network-exploitable; for Ubuntu, MacOS, and Solaris, the percentage of network vulnerabilities is over 50%; the fewest percentages of network exploitable vulnerabilities have been detected in Windows (46%) and Novell (40%). Ubuntu and Red Hat have the highest number of network-exploitable vulnerabilities (618 and 420 vulnerabilities correspondingly). Practically this means, that Ubuntu and Red Hat used "from the box" without a proper security configuration (e.g., installing a firewall, antivirus software, and reducing a surface of attack by stopping unused network services) have a higher chance to be hacked from the Internet than other OSs.

Another information of concern is the significant number of vulnerabilities (from 88% to 98% for different OSs) that do not require user authentication to be exploited. It means that most of the hacker attacks would simply bypass built-in OS access control mechanisms making them useless.

Note here that the sum of vulnerabilities within the CVSS "impact" metric group is higher than the total number of disclosed vulnerabilities presented in Table V. This is explained by the fact that once exploited, most of vulnerabilities would allow an attacker to compromise at once all system security properties: confidentiality, integrity, and availability.

E. Interdependency Between Vulnerability Severity and Days-of-Grey Risk

Any software user would expect that vendors always try to fix the most severe vulnerabilities first. On the other hand, a rational vendor would take a risk-based view to decide which vulnerability to give high priority by taking into account the likelihood of exploit.

A vulnerability may be difficult to exploit (e.g., requires a very high competence or simply security controls commonly used make exploitation very difficult). Ignoring the likelihood of exploits from the vendor's point of view may be a recipe for wasting resources. The CVSS *base score* can be considered as a good risk-based indicator as it integrates both *vulnerability impact metrics* (impact on integrity, confidentiality, and availability)

TABLE V
CVSS-BASED VULNERABILITY STATISTICS

Year	CVSS criteria	Vulnerabilities	Ubuntu	Windows	Red Hat	Novell	MacOS	Solaris
Inherited	Attack vector	Local	6	0	28	19	0	6
		Adj. network	2	0	4	1	0	0
		Network	7	0	14	6	0	7
	Auth.	Required	1	0	1	1	0	2
		None	14	0	45	25	0	11
	Impact	Confidentiality	7	0	15	12	0	7
		Integrity	7	0	12	7	0	3
		Availability	12	0	39	20	0	9
2012	Attack vector	Local	25	1	14	24	1	26
		Adj. network	4	0	1	4	0	12
		Network	35	9	16	4	1	9
	Auth.	Required	9	0	3	4	0	7
		None	55	10	28	28	2	40
	Impact	Confidentiality	35	9	12	20	2	16
		Integrity	25	9	17	13	0	18
		Availability	45	8	24	25	0	39
2013	Attack vector	Local	109	31	34	88	27	21
		Adj. network	11	1	3	11	1	0
		Network	76	27	34	25	32	10
	Auth.	Required	24	2	7	12	3	5
		None	172	57	64	112	57	26
	Impact	Confidentiality	120	47	48	72	39	7
		Integrity	97	40	42	45	34	11
		Availability	140	51	49	90	31	27
2014	Attack vector	Local	81	26	14	72	13	22
		Adj. network	4	4	1	3	0	1
		Network	103	34	57	54	27	14
	Auth.	Required	11	3	1	15	0	3
		None	177	61	71	114	40	34
	Impact	Confidentiality	101	52	54	70	40	20
		Integrity	79	44	50	54	35	14
		Availability	144	44	60	106	34	30
2015	Attack vector	Local	32	105	14	12	9	31
		Adj. network	0	1	1	1	0	0
		Network	219	72	147	39	5	43
	Auth.	Required	41	4	48	4	0	5
		None	210	174	114	48	14	69
	Impact	Confidentiality	147	158	95	37	12	31
		Integrity	154	140	90	38	14	34
		Availability	203	136	134	40	12	63
2016	Attack vector	Local	65	85	9	12	9	3
		Adj. network	0	16	1	0	0	0
		Network	149	103	115	11	39	1
	Auth.	Required	7	27	2	0	0	0
		None	207	177	123	23	48	4
	Impact	Confidentiality	114	173	98	20	42	4
		Integrity	103	123	94	19	26	4
		Availability	178	134	99	22	35	4
2017	Attack vector	Local	21	132	27	19	3	3
		Adj. network	2	0	2	2	1	0
		Network	6	58	24	4	37	2
	Auth.	Required	0	8	1		1	0
		None	29	182	52	25	40	5
	Impact	Confidentiality	20	181	37	17	38	1
		Integrity	21	81	37	18	27	3
		Availability	27	87	46	23	29	3
Total	Attack vector	Local	339	380	140	246	62	112
		Adj. network	23	22	13	22	2	13
		Network	595	303	407	143	141	86
	Auth.	Required	93	44	63	36	4	22
		None	864	661	497	375	201	189
	Impact	Confidentiality	544	620	359	248	173	86
		Integrity	486	437	342	194	136	87
		Availability	749	460	451	326	141	175

determining vulnerability severity and *exploitability metrics* (attack vector, access complexity, and needs for authentication) that define the likelihood of exploitation (https://nvd.nist.gov/vuln-metrics/cvss/v2-calculator).

The box-and-whisker diagrams in Fig. 7 show the numbers of days-of-grey-risk corresponding to vulnerabilities of different CVSS scores. They allow us to compare how quick OS vendors fix the least (CVSS severity score is in the range [1.0–3.0]) and the most (CVSS severity score is in the range [8.0–10.0]) severe vulnerabilities.

Unfortunately, it is shown that the days-of-grey-risk metric does not actually depend on the CVSS vulnerability severity rating. The presented results disprove a widespread hypothesis that software vendors put more effort into fixing the most critical vulnerabilities. To some extent it seems to be true for the Red Hat OS. The Windows team spends approximately the same time on fixing the most and the least severe vulnerabilities (127 versus 128 days on average). However, the developers of other OSs spend considerably more time on fixing critical vulnerabilities as compared to the least severe ones.

F. Most Critical Types of OS Vulnerabilities

NVD classifies all vulnerabilities using the CWE scheme. CWE is a formal list of software weakness types proposed by the MITRE Corporation (https://cwe.mitre.org/).

Our analysis demonstrates that the most numerous types of vulnerabilities for OSs in general are as follows:

1) CWE-119 (24%)—Improper restriction of operations within the bounds of a memory buffer caused by weaknesses of certain programming languages (often C and C++) that do not control bounds for the memory buffer that is being addressed. Vulnerabilities of this type usually cause arbitrary code execution, altering the intended control flow, and leading to accesses to protected information or system crash.
2) CWE-264 (23%)—Weaknesses and implementation mistakes in permissions, privileges, and access control.
3) CWE-200 (15%)—Intentional or unintentional information exposure to an actor that is not explicitly authorized to have access to that information.
4) CWE-20 (13%)—Improper input validation that may result in altered control flow, arbitrary code execution, or illegal access to and control of resources.
5) CWE-399 (6%)—Improper management of system resources, e.g., memory allocation or reallocation.
6) CWE-189 (5%)—Numeric errors related to improper calculation or conversion of numbers.
7) CWE-362 (2%)—Concurrent code execution using a shared resource with improper synchronization also known as a *Race Condition*.
8) CWE-310 (2%)—Cryptographic issues including missing encryption of sensitive data or key management errors.
9) CWE-94 (1%)—Improper control of code generation also known as *Code Injection* that often happens when software allows a user's input to contain code syntax.

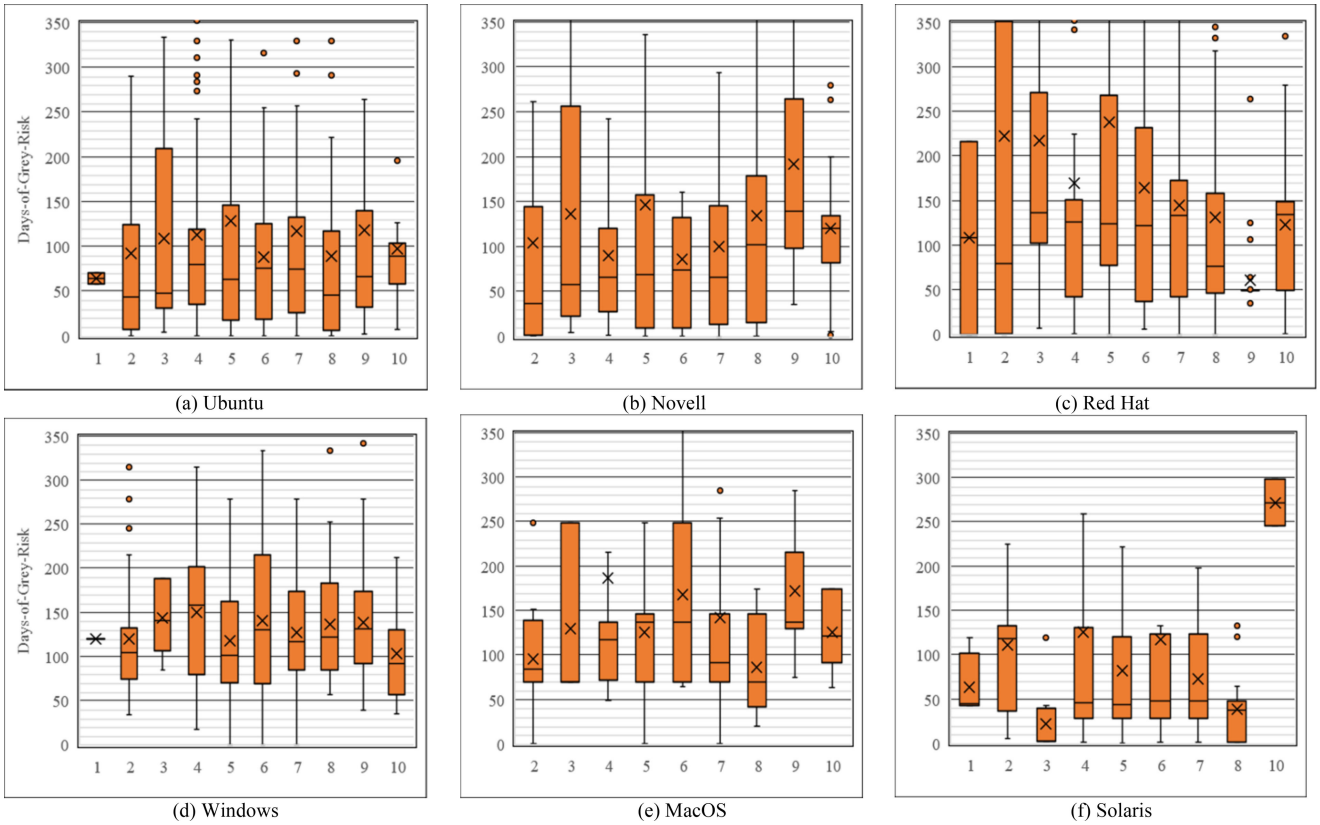

Fig. 7. Box-and-whisker diagrams showing a days-of-grey-risk statistics (Y-axes) for vulnerabilities of different CVSS severity scores (X-axes). (a) Ubuntu. (b) Novell. (c) Red Hat. (d) Windows. (e) MacOS. (f) Solaris.

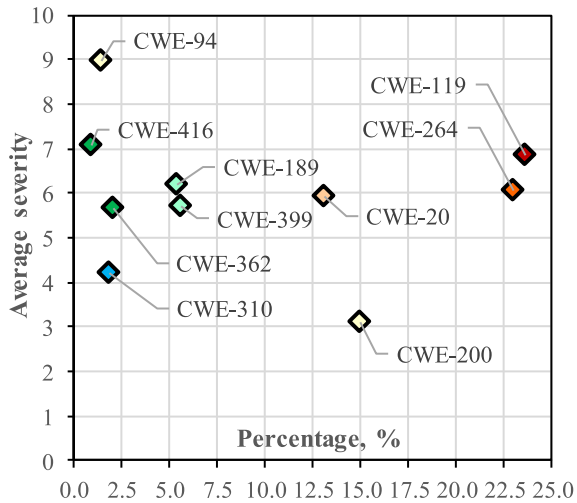

Fig. 8. Most numerous vulnerability types and their severity.

10) CWE-416 (1%)—The use after free vulnerabilities that results in referencing memory after it has been freed, which can cause a program to crash, use unexpected values, or execute code.

Analyzing both the quantity and CVSS severity scores of vulnerabilities of different type (see Fig. 8), we can conclude that the most critical ones are: CWE-119, CWE-264, and CWE-20. CWE-94, despite its small frequency, has the maximum severity on average (8.9).

Our analysis shows that CWE-119 vulnerabilities, also widely known as buffer overflow, still remain the most dominating and severe security flaws for all OSs. On one hand, this can be explained by the fact that most OSs, written in C/C++, are prone to this type of weakness. On the other hand, it points to the fact that programmers either rarely pay enough attention to such widely known problems that have been around for years, and/or follow best software development practices, and/or make use of numerous techniques proposed to cope with the buffer overflow issue.

As a result, vulnerabilities of the CWE-119 type (e.g., CVE-2016-7277, CVE-2016-4658, or CVE-2016-4598) often allow remote attackers to execute arbitrary code, read protected data, or cause a denial of service.

Distribution of different types of vulnerabilities for particular OSs can be found in [13].

G. Common OS Vulnerabilities

This section examines the vulnerabilities discovered in more than one OS by analyzing CPE entries assigned to them. They are usually called common or shared [23], [24]. With careful planning, an adversary may exploit different vulnerabilities and achieve (a nearly) simultaneous compromise of heterogeneous OSs. However, common vulnerabilities make compromising installations with heterogeneous OSs much easier and, once exploited, can cause a global epidemic of cyberattacks. They exist due to inheriting considerable parts of the OS code from its

predecessor or reusing common components (system libraries, third party software components, OS kernels, etc.).

The common vulnerabilities are most often discovered in different releases of the same OS or in a family of related OSs, e.g., BSD Unix (OpenBSD, FreeBSD, NetBSD) or Linux (Red Hat, CentOS, Novell, Ubuntu), etc.

For example, our analysis shows that 62 out of 63 (98%!) vulnerabilities reported by the NVD database in the most recent Apple MacOS 10.13 were also found in MacOS 10.8. The percentages of vulnerabilities shared between Microsoft Windows Server 2012 and its 2016 version are equal to 76% (123 vulnerabilities out of 165 found in Windows Server 2016 by the end of December 2017). It is remarkable that this number also includes 114 vulnerabilities (69%) that Windows Server 2016 shares with Windows Server 2008 and 23 vulnerabilities (14%) shared with Windows Vista. Moreover, at least six vulnerabilities in the SMB protocol, causing this year a massive WannaCry cyber attack, are traced to Windows Server 2003 and even to Windows XP.

The 6.x and 7.x (last updated on 01.08.2017) versions of Red Hat Enterprise Linux, and the 12.4 and 16.4 (released on 21.04.2016) versions of Ubuntu Server share up to 75% and 70% of common vulnerabilities correspondingly. It is also worth noting that Oracle Solaris 11.3 in 2016 shared 29% of vulnerabilities with Oracle Solaris 10.0 and 24% with Oracle Linux 7 but none with the 11.0 version, analyzed in this paper.

These results confirm that the developers of OSs reuse significant pieces of code from previous releases without really analyzing their vulnerability or improving their security.

Sometimes hackers and security analysts discover vulnerabilities that are common for even different OS families. One such vulnerability is CVE-2008-4609 found in October, 2008. It caused a denial-of-service attack for a variety of OSs and their versions, including Linux, BSD Unix, Microsoft Windows, Cisco IOS, and possibly many others [41], [42]. The vulnerability manipulated the state of transmission control protocol (TCP) connections exploiting an algorithmic error in protocol implementation in various OSs. A remote attacker was able to cause connection queue exhaustion by flags manipulation in the TCP header of crafted network packets sent to a victim-computer.

Fig. 9 shows common vulnerabilities correlated between Linux and Unix OSs during 2012–2017 (Windows did not share any vulnerabilities with the rest of the studied OSs). Eighty-five were disclosed in all three Linux OSs (Ubuntu, Novell and Red Hat) and ten were shared between Red Hat, Ubuntu, and Solaris. In addition, there were six groups of vulnerabilities shared between different OS pairs: Ubuntu and Novell—245, Red Hat and Ubuntu—60, Novell and Red Hat—36; Red Hat and MacOS—245; Red Hat and Solaris—4; Ubuntu and Solaris—5. These data emphasize the importance on analyzing the vulnerabilities of diverse OSs.

The numbers in brackets in Fig. 9 correspond to those vulnerabilities observed in Linux kernels (we counted the vulnerabilities in Linux kernels based on their own CPE identifiers that are different from those assigned to Linux OSs themselves, as discussed in Section II-C). Thus, Fig. 9 clearly demonstrates that the largest number of common and group vulnerabilities shared

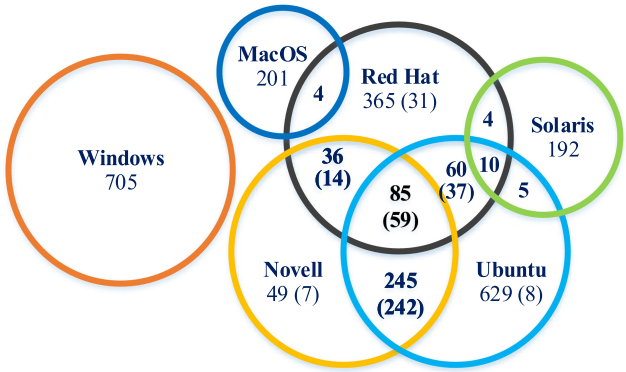

Fig. 9. Number of individual and common vulnerabilities shared by different OSs.

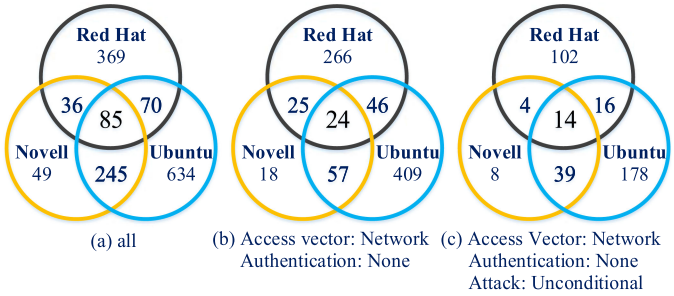

Fig. 10. Number of individual and common vulnerabilities shared by Linux OSs depending on the attack vector.

between the Ubuntu, Novell, and Red Hat OSs are those discovered in the Linux kernels (versions 3.2.x, 3.0.x, and 2.6.32) used by them. In total, the percentage of common vulnerabilities shared between the three Linux OSs varies from 8% (for the 3-version system) to almost 45% (for the 2-version systems combining Ubuntu and Novell).

It is also noteworthy that the two Unix OSs, Solaris and MacOS, do not have known common vulnerabilities at all while they share certain numbers of vulnerabilities with different Linux OSs.

The number of vulnerabilities shared by two or more OSs, can be used as a measure of diversity between them [23]. Software diversity [14], [18], [21] has been used as a major fault and intrusion tolerance mechanism to design safety-critical computer systems. Thus, choosing the most diverse OSs allows the creation of the most secure and reliable multiversion system.

Our empirical study demonstrates that the NVD database can help in determining the most diverse software products by using CPE identifiers. At the same time, results reported in [23] might be further verified as the authors may not have considered all common and group vulnerabilities observed in Linux kernels (the manual vulnerability classification performed by Garcia *et al.* [23] might be less accurate than our approach that relies on using CPE identifiers assigned to Linux kernels).

Fig. 10 shows a distribution of individual, common, and group vulnerabilities in Linux OSs depending on the attack surface. A Venn diagram on Fig. 10(b) presents only remotely exploitable

vulnerabilities. This type of vulnerability is the more severe, as defined by the CVSS scoring system [40].

In addition, remotely exploitable vulnerabilities can be split into the following two major subgroups depending on the attack execution initial conditions [43]:

1) on attack object request or event;
2) unconditional.

In the first case, the attack is triggered by a user of a vulnerable system. For example, a user can access a remote web site with crafted content (a crafted image or JavaScript code, a malformed XML/PDF file, embedded font, etc.) that would trigger a buffer overflow in a vulnerable web browser, system library or component (e.g., video decoder, file parser, etc.). Thus, these vulnerabilities are more relevant to desktop PCs.

Vulnerabilities identified as *unconditional* allow remote attackers to initiate the attack via sending crafted packets to a vulnerable system service or application daemon. They include vulnerabilities in the network protocol stack, which is a part of the OS kernel; basic OS network daemons (ssh, samba, nfs, etc.) and application-level network services (web-server, FTP-server, etc.).

Undoubtedly, unconditional vulnerabilities have higher severity and are of great importance for network servers. Unfortunately, the CVSS scoring system used by NVD does not distinguish between these two groups of remotely executable vulnerabilities.

In Fig. 10(c), we report the numbers of unconditional remotely exploitable vulnerabilities in different Linux OSs based on a manual classification.

IV. Using OS Diversity to Improve System Security and Intrusion Tolerance

A. OS Diversity and Intrusion Tolerance Architecture

Software vulnerabilities represent threats to dependability and, in particular, to security, that are additional to faults, errors, and failures, traditionally dealt with by the dependability community [44], [45]. Design diversity is one of the most efficient methods for providing software fault-tolerance [14], [15] and improving dependability.

Often, researchers consider vulnerabilities as a special case of software faults activated by an attacker [45]. As a result, many studies focus on applying diversity to boost the intrusion tolerance of a system in the same way as software design diversity is used to ensure fault-tolerance.

In general, a diverse computer system consists of two or more replicas that run diverse software. The main assumption behind software diversity is that designs and implementations developed independently (programmed by different teams, using diverse languages and development methodologies) will exhibit failure and vulnerability diversity.

Diversity, being a part of the intrusion tolerance mechanism, can improve system security, especially availability [23], [24], [46], [47]. However, the impact of software diversity on system confidentiality and integrity taking into account common vulnerabilities and the dynamic process of vulnerability discovery and patching is less understood.

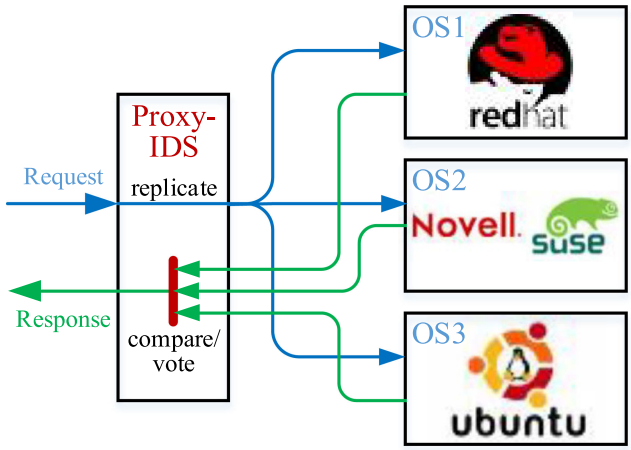

Fig. 11. Intrusion-tolerant architecture under study.

There have been an increasing number of approaches and architectures proposed to build intrusion-tolerant systems. They employ different techniques to tolerate intrusions: adaptive redundancy and diversification principles [48], [49], asynchronous replications [50], [51] and Byzantine agreement protocols [52], [53], replica "cleansing" [54], etc.

In this paper, we consider only one of many possible intrusion-tolerant architectures coping with vulnerabilities of OSs. This architecture, shown in Fig. 11, comprises functionally redundant servers running diverse OSs and a proxy/IDS that mediates client requests to all servers and also verifies their behavior, as described in [46], [48], and [55]. It is well suited to tolerating intrusions in replicated server systems, e.g., in building intrusion-tolerant web servers.

Proxy is the only single component of the architecture accessible directly to a client. It forwards client requests to the replicas without actually processing them. Thus, it is simple enough to be considered as secure [55]. Intrusions are detected through the comparison of the server outputs before returning the result to clients.

OSs of different families (e.g., Unix, Linux, Windows, MacOS) are more diverse, by nature, than those belonging to the same OS family. Indeed, Fig. 9 shows that there are OSs, which do not share known vulnerabilities.

Thus, it is possible to pick out the following configurations of a three-version diverse system that do not have common and group, i.e., shared by any two OSs, vulnerabilities at all (sorted by the total number of vulnerabilities, which are shown in brackets).

1) Novell, MacOS, Solaris (831);
2) Windows, MacOS, Solaris (1121);
3) Windows, MacOS, Novell (1325);
4) Windows, Novell, Solaris (1331);
5) Windows, MacOS, Ubuntu (1944).

However, using OSs of different families for building a diverse intrusion-tolerant system might cause various compatibility, portability, and synchronization issues. This is why developers of diverse intrusion- and fault-tolerance systems often opt for using OSs of the same family [56]–[58]. Moreover, the

absence of known common vulnerabilities does not guarantee that there are no hidden vulnerabilities shared between diverse software.

In this section, we do not focus on finding the most diverse configuration among studied OSs. Instead, our aim is to examine a common case scenario that assumes existence of common and group vulnerabilities. In the rest of this section, we analyze how these vulnerabilities affect different security attributes (availability, integrity, and confidentiality) of a diverse system. To quantify our study, we chose a diverse intrusion-tolerant architecture comprising the three Linux-based OSs (Ubuntu, Novell, and Red Hat), whose common remotely exploitable unconditional vulnerabilities were studied in Section III-G.

B. The Threat Model and Assumptions

In the proposed intrusion-tolerant architecture (Fig. 11), all user requests and server responses synchronously pass through the proxy. The intrusion detection algorithm assumes that all noncompromised servers give the same answer to the same request [47], [48]. Thus, an intrusion is detected when the outputs are different due to an exploited vulnerability in one of diverse OSs. Majority voting is then used to identify a suspicious replica, isolate, cleanse/repair, and reinsert it without interrupting the service. The general assumptions, which follow from the architecture description, are as follows.

1) Diverse replicas can be maintained either by using a suitable replication protocol (e.g., a Byzantine agreement protocol that exploits trusted components [53]) or in the execution layer of an intrusion-tolerant system (e.g., in [59], any protocol can be used to replicate commands, but the service itself is executed in just three replicas implementing majority voting).
2) The system implements graceful degradation in such a way that one compromised replica can breach system confidentiality (as the state is exposed); two compromised replicas can breach system integrity (as majority voting does not work anymore), and three compromised replicas breach system availability (as there are no correct replicas to provide the service).
3) An attacker cannot directly interact with any individual replica; all requests and responses go via the proxy.
4) An attacker has only "one shot" at compromising the whole replicated system; a compromised replica, detected by IDS, is cleansed before an attacker will get a chance to compromise other replica(s) [47].

It follows from the above assumptions that if diverse OSs do not have common vulnerabilities (i.e., they are 100% diverse), a hacker would not be able to compromise all replicas at the same time with a single malicious request. Thus, from the availability point of view, the least vulnerable diverse configuration is the one with the minimum number of such vulnerabilities. However, a multiversion architecture enlarges the *overall attack surface* (i.e., the total number of vulnerabilities that can be exploited) and, hence, it could weaken other system security attributes, such as confidentiality or integrity [21].

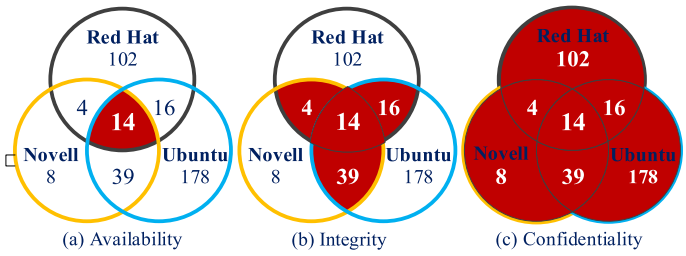

Fig. 12. Venn diagrams showing attack surface of the three-version intrusion-tolerant system.

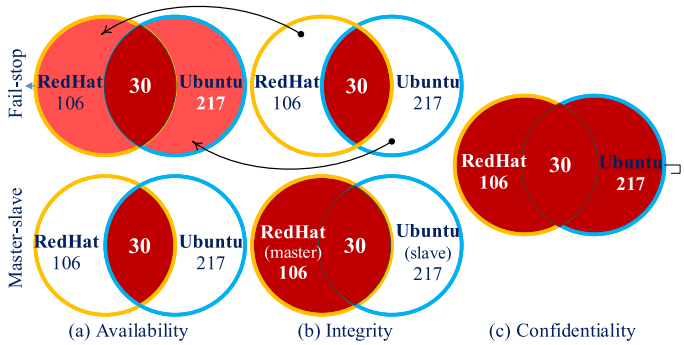

Fig. 13. Venn diagrams showing attack surface of the two-version intrusion-tolerant system.

Our threat model considers attack surfaces of a replicated diverse system for different types of attacks targeting availability, integrity, and confidentiality in the following ways:

1) The three-replicated system preserves *availability* if at least one replica remains available (i.e., 1-out-of-3 replicas returns a response); thus, to make the system unavailable, an attacker needs to target those vulnerabilities, common to all replicas, which impact availability [see Fig. 12(a)]; attacking any other vulnerability would not make the entire diverse system unavailable.
2) The three-replicated system preserves *integrity* if 2-out-of-3 (the quorum) replicas return the correct response; thus, to compromise system integrity an attacker needs to target those vulnerabilities, common for any two replicas, which impacts *integrity* [see Fig. 12(b)].
3) Compromising any of the diverse OSs would break system confidentiality; thus, an attacker can target any vulnerability of any replica, which impacts confidentiality [see Fig. 12(c)].
4) The attack surface of the 2-replicated diverse system has some differences depending on the system implementation [see Fig. 13].
5) If a system is designed/configured to stop its operation once it detects data discrepancy (i.e., a fail-stop system [60]), an attack compromising integrity of 1-out-of-2 replica would make the whole system unavailable.
6) If one of the OS versions is considered to be more trusted (a master replica), the system, when it detects inconsistency, will continue its operation using data provided by the more trusted master OS; a similar approach was used

TABLE VI
CRITICAL ATTACK SURFACES FOR DIFFERENT SECURITY PROPERTIES IN VARIOUS DIVERSE CONFIGURATIONS

System architecture	OS			No of vulnerabilities		
	Ubuntu	Novell	Red Hat	Availability	Integrity	Confidentiality
Single-version	*			183	124	128
		*		**56**	**34**	**37**
			*	98	69	73
Multi-version	*	*		45	**34**	137
	*		*	23	69	184
		*	*	**13**	**34**	**98**
	*	*	*	**10**	35	201

in the HACQIT project [55], [61]; in our study, we assume that all two-version architectures are configured as master-slave; OS having the least number of discovered vulnerabilities is considered as the master replica.

Figs. 12 and 13 quantify attack surfaces of different security attributes using the number of individual and common network vulnerabilities discovered in Linux-based OSs, as reported in Section III-G [see Fig. 10(c)]. A critical part of an attack surface, marked in red, includes those groups of vulnerabilities, which would allow a remote attacker to compromise the whole system (i.e., to breach a certain security property) with the single malicious request.

C. Examining Static and Dynamic Impact of OS Diversity on Availability, Confidentiality, and Integrity of the Intrusion-Tolerant System

In this section, we quantitatively examine the vulnerability of several possible configurations of the intrusion-tolerant architecture discussed above. As intrusion-tolerant servers are usually used to provide critical network services, we consider only remotely exploitable unconditional vulnerabilities (see Section III-G for more details). Table VI quantifies the critical attack surface for individual OSs and various configurations of a diverse intrusion tolerant system taking into account vulnerability impact on different security properties, as defined by the CVSS impact score.

It is clear that developers of intrusion-tolerant systems deploying OS diversity have to tradeoff between different security properties.

Fig. 14 demonstrates the interplay between a number of vulnerabilities affecting availability and confidentiality [see Fig. 14(a)], and availability and integrity [see Fig. 14(b)] for individual OSs and diverse configurations.

If one is ready to sacrifice confidentiality in favour of availability, the three-version architecture is the best choice. It also provides a good compromise between availability and integrity. The pair Novell and Red Hat seems to be the best diverse configuration for improving all security attributes. It has the least number of vulnerabilities targeting integrity and confidentiality and also provides a good compromise with availability.

Among the individual, OSs Novell has the smallest number of remotely exploitable vulnerabilities impacting availability,

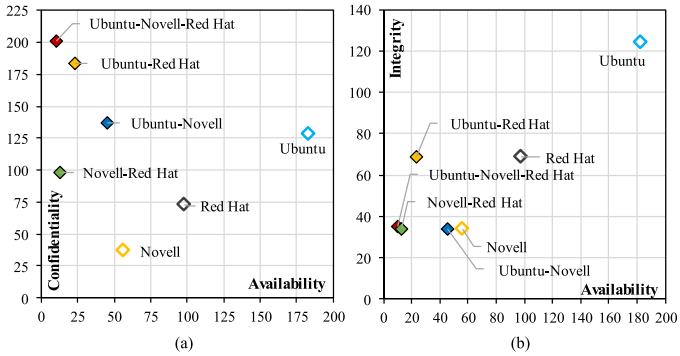

Fig. 14. Tradeoffs between vulnerabilities impacting. (a) Availability and confidentiality. (b) Availability and integrity.

TABLE VII
SUMMARY OF FOREVER-DAY VULNERABILITY STATISTICS (ATTACK SURFACE) FOR VARIOUS DIVERSE OS CONFIGURATIONS

Operating System		Diverse system configurations			
Ubuntu		*	*		*
Novell		*		*	*
Red Hat			*	*	*
Availability attack surface					
No of vulnerabilities per day	avg.	2.81	2.87	0.61	0.48
	min	0	0	0	0
	max	11	6	4	4
No of vulnerability free days		346	238	1380	1401
No of days with the least number of forever-day vulnerabilities		790	693	2034	2192
Integrity attack surface					
No of vulnerabilities per day	avg.	2.34	5.91	2.34	2.38
	min	0	0	0	0
	max	11	14	11	11
No of vulnerability free days		487	191	487	193
No of days with the least number of forever-day vulnerabilities		1590	364	1590	1301
Confidentiality attack surface					
No of vulnerabilities per day	avg.	7.99	13.46	8.98	13.92
	min	0	3	1	3
	max	30	47	29	47
No of vulnerability free days		47	0	0	0
No of days with the least number of forever-day vulnerabilities		1020	101	1167	0

integrity, and confidentiality. At the same time, Ubuntu should not be considered as a good choice in any scenario.

A more optimal decision regarding the best diverse configuration of on intrusion-tolerant system can be made dynamically by considering how many common vulnerabilities existed each day in a particular configuration (see Figs. 15–17).

Table VII summarises the statistics shown in Figs. 15–17 and provides arguments for and against each diverse configuration.

As expected, the three-version system significantly reduces a surface of network attacks targeting availability down to 0.48 vulnerabilities per day on average. It maintained the lowest number of forever-day vulnerabilities during the whole six-year period, during which 1401 were days with no known vulnerabilities at all.

The combination of Novell and Red Hat is the best diverse configuration for a system whose top priority is availability. On average, it exhibits 0.61 vulnerabilities per day and ensures

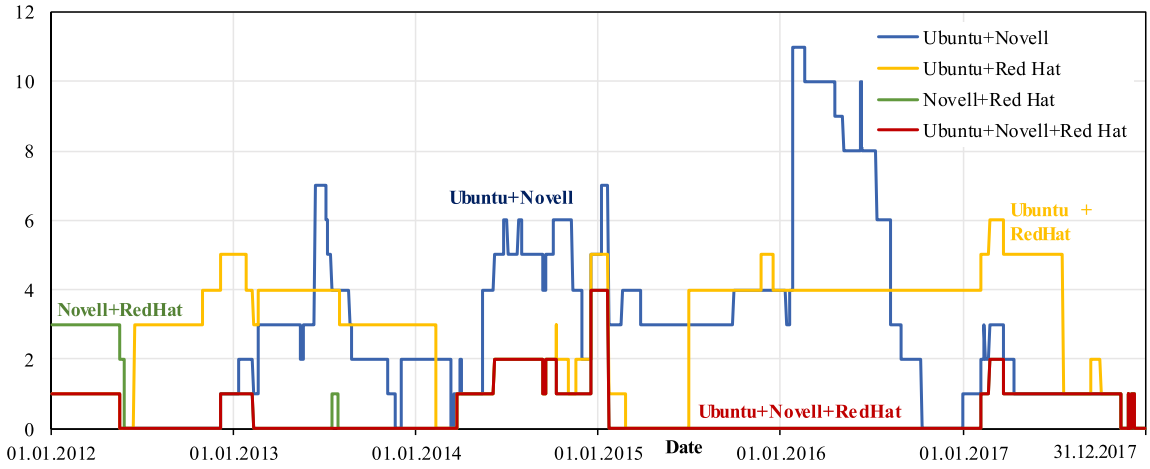

Fig. 15. Forever-day vulnerabilities in different configurations of a diverse intrusion tolerance system affecting availability.

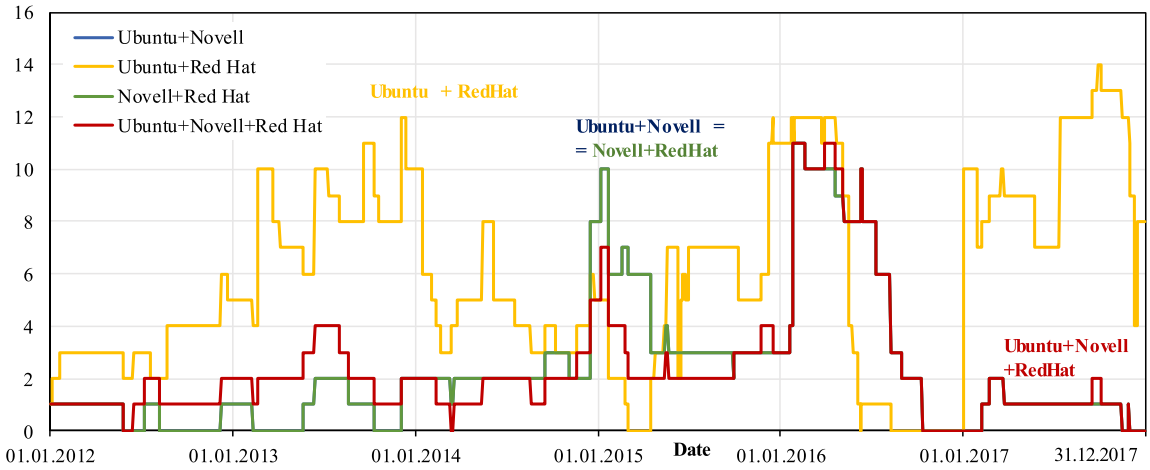

Fig. 16. Forever-day vulnerabilities in different configurations of a diverse intrusion tolerance system affecting integrity.

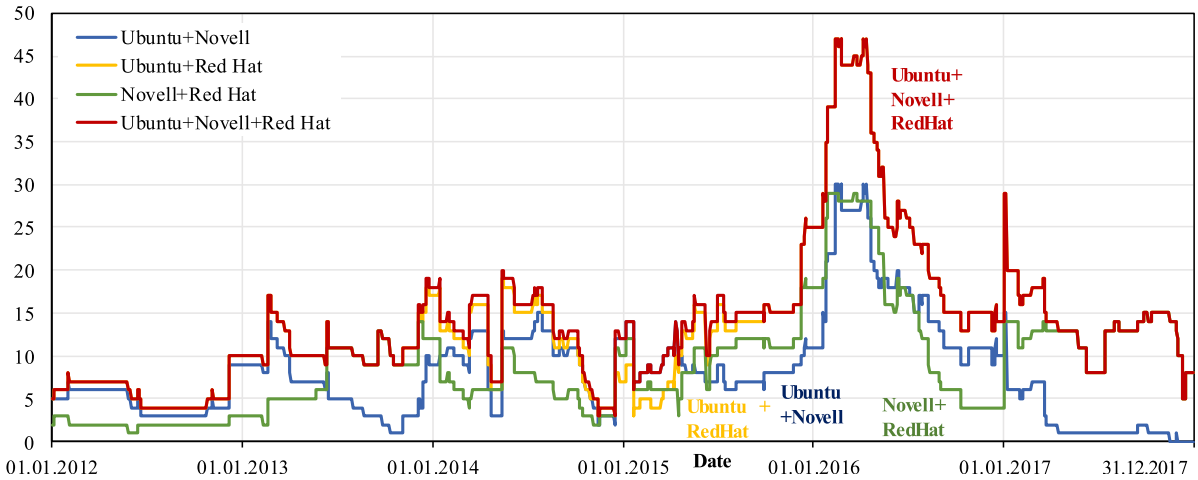

Fig. 17. Forever-day vulnerabilities in different configurations of a diverse intrusion tolerance system affecting consistency.

almost the same number of vulnerability-free days (1380) as the three-version system.

However, the three-version system is not the best configuration for integrity-critical applications.

It exhibits, on average, 2.38 vulnerabilities per day, which is slightly worse than the pairs of Ubuntu–Novell and Novell–Red Hat (i.e., 2.34), and had considerably fewer vulnerability-free days (193 versus 487).

Finally, a diverse system, for which the most important security property is confidentiality, would benefit from using either the combination of Ubuntu and Novell, or Red Hat and Novell. The former had the least average number of forever-day vulnerabilities per day (7.99) affecting confidentiality and ensured 47 vulnerability-free days. However, the latter exhibited the lowest number of forever-day vulnerabilities during the longer period (1167 versus 1020 days).

The three-version configuration is not recommended for use in confidentiality-critical systems as it significantly enlarges an attack surface up to 13.92 vulnerabilities per day on an average.

V. Conclusion

A significant growth of the total number of vulnerabilities discovered in modern OSs as well as the general tendency towards increasing their severity demonstrate the serious security challenges and risks that OS developers and users face.

It is very important to understand that the crucial parameters affecting system security were not only the total number of vulnerabilities disclosed in a particular software product and their severity, but also so called *days-of-grey-risk*, which show how fast software vendors issue patches fixing disclosed vulnerabilities, and a number of *forever-day vulnerabilities* defining the attack surface.

Our analysis showed that the average days-of-grey-risk for the studied OSs varied from 89 days for Ubuntu up to 130 days for Red Hat. In addition, it was found that on average 28 forever-day vulnerabilities for the investigated OSs existed every day during 2012–2017 (a number of such vulnerabilities varies on average between 8 for Solaris and 48 for Ubuntu).

Thus, this paper clearly supported the claim that decreasing days-of-grey-risk and reducing a number of forever-day vulnerabilities was one of the main challenges in improving security of OSs.

It is worrying that as our study shows, the rate at which OS developers issue security updates in general does not depend on vulnerability severity. Average days-of-grey-risk for the most critical vulnerabilities remains 24% higher than that calculated for vulnerability of the lowest severity.

Another important finding was that developers reuse significant pieces of code from previous releases (which was not surprising itself) without really analyzing their vulnerability and improving their security. Moreover, buffer overflow vulnerabilities still remain the most dominant and severe security flaws for all OSs despite many techniques being proposed to cope with this type of vulnerability.

Thus, our findings demonstrated worrying shortcomings in the engineering practices and policies for developing security updates adopted by OS vendors, as well as, in the maintenance management processes they run.

Another specific aspect that this paper studied was the vulnerabilities that were discovered in more than one OS. Such vulnerabilities, common to different OSs and even different OS families, could lead to large-scale hacker attacks and virus epidemics.

This called for the application of specially tailored techniques for intrusion tolerance. One such was based on adopting software diversity. In this paper, we quantitatively analyzed how OS diversity impacts attack surface taking into account individual and common vulnerabilities.

Unlike other studies, we had investigated how diversity affects various security attributes: availability, integrity, and confidentiality using historical statistics from the CVE and NVD vulnerability databases. We confirmed that the more OS versions we used and the more diverse they were, the more the system became tolerant to attacks targeting system availability. However, diversity could undermine integrity and confidentiality properties by enlarging the system attack surface.

In particular, in this paper, we considered different possible configurations of two- and three-version intrusion-tolerant systems built by combining Linux-based OSs: Ubuntu, Novell, and Red Hat. Our practical findings based on real vulnerability statistics confirmed that a three-version architecture was the best choice to ensure high system availability. On average, it maintained only 0.48 forever-day vulnerabilities targeting system availability. Correspondingly, it gave better average results, by a factor of 10.3, than those for individual OSs and 4.5 times better than two-version systems.

However, for the three-version system, the number of forever-day vulnerabilities targeting data confidentiality was 3.8 times higher. It was fair to note that even the best two-version configuration (Ubuntu+Novell) enlarges the confidentiality attack surface by a factor of 2.2. These results showed that OS diversity in certain scenarios (e.g., when availability and/or confidentiality are the top-most priority) definitely improved system intrusion tolerance. However, this was not a panacea for intrusion targeting confidentiality. This calls for developing more effective security mechanisms in addition to traditional intrusion-tolerant solutions.

Reference

[1] B. Clark, "Hackers take hospital offline, demand $3.6m ransom," 2016. [Online]. Available: http://thenextweb.com/insider/2016/02/15/hackers-take-hospital-offline-demand-3-6m-ransom/

[2] C. Williams, "Passengers ride free on SF Muni subway after ransomware infects network, demands $73k," 2016. [Online]. Available: http://www.theregister.co.uk/2016/11/27/san_francisco_muni_ransomware/

[3] "Investigation: WannaCry cyber attack and the NHS," Nat. Audit Office, London, U.K, 2017.

[4] L. H. Newman, "The biggest cybersecurity disasters of 2017 so far," Jul. 1, 2017. [Online]. Available: https://www.wired.com/story/2017-biggest-hacks-so-far/

[5] MITRE Corporation, "Common vulnerabilities and exposures. Terminology," 2017. [Online]. Available: https://cve.mitre.org/about/terminology.html

[6] Microsoft Inc., "Description of software update services and windows server update services changes in content for 2016," 2016. [Online]. Available: https://support.microsoft.com/en-us/help/3215781/description-of-software-update-services-and-windows-server-update-services-changes-in-content-for-2016

[7] J. Jones, "Days-of-risk in 2006: Linux, Mac OS X, Solaris and Windows," 2006. [Online]. Available: http://www.csoonline.com/article/2136935/data-protection/days-of-risk-in-2006—linux–mac-os-x–solaris-and-windows.html

[8] P. Edmonds, "When it comes to protection from vulnerabilities, process trumps 'Many Eyes,'" 2007. [Online]. Available: https://technet.microsoft.com/en-us/library/cc512608.aspx

[9] A. Patrizio, "Report says Windows gets the fastest repairs," 2007. [Online]. Available: http://www.internetnews.com/security/article.php/3667201

[10] J. Reavis, "Linux vs. Microsoft: Who solves security problems faster?" 2000. [Online]. Available: http://www.reavis.org/research/solve.shtml

[11] J. Jones, "Basic guide to days of risk," 2007. [Online]. Available: http://www.csoonline.com/article/2136934/data-protection/basic-guide-to-days-of-risk.html

[12] D. Goodin, "Rise of 'forever day' bugs in industrial systems threatens critical infrastructure," 2012. [Online]. Available: http://arstechnica.com/business/2012/04/rise-of-ics-forever-day-vulnerabiliities-threaten-critical-infrastructure/

[13] A. Gorbenko, A. Romanovsky, O. Tarasyuk, and O. Biloborodov, "Experience report: Study of vulnerabilities of enterprise operating systems," in *Proc. IEEE 28th Int. Symp. Softw. Rel. Eng.*, Toulouse, France, 2017, pp. 205–215.

[14] A. Avizienis, "The N-version approach to fault-tolerant software," *IEEE Trans. Softw. Eng.*, vol. SE-11, no. 12, pp. 1491–1501, Dec. 1985.

[15] A. Avizienis and J.-C. Laprie, "Dependable computing: From concepts to design diversity," *IEEE Proc.*, vol. 74, no. 5, pp. 629–638, May 1986.

[16] B. Littlewood, P. Popov, and L. Strigini, "Design diversity: An update from research on reliability modelling," in *Proc. 9th Saf.-Critical Syst. Symp.*, Bristol, U.K., 2001, pp. 139–154.

[17] A. Avizienis and L. Chen, "On the implementation of N-version programming for software fault tolerance during execution," in *Proc. IEEE Comput. Softw. Appl. Conf.*, Kharagpur, India, 1977, pp. 149–155.

[18] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Softw. Eng.*, vol. SE-1, no. 2, pp. 221–232, Jun. 1975.

[19] P. Popov, "Models of reliability of fault-tolerant software under cyber-attacks," in *Proc. IEEE 28th Int. Symp. Softw. Rel. Eng.*, Toulouse, France, 2017, pp. 228–239.

[20] J. Dobson and B. Randell, "Building reliable secure computing systems out of unreliable insecure components," in *Proc. IEEE Conf. Secur. Privacy*, Oakland, CA, USA, 1986, pp. 187–193.

[21] B. Littlewood and L. Strigini, "Redundancy and diversity in security," in *Proc. 9th Eur. Symp. Res. Comput. Secur.*, 2004, pp. 423–438.

[22] I. Gashi, A. Povyakalo, L. Strigini, M. Matschnig, T. Hinterstoisser, and B. Fischer, "Diversity for safety and security in embedded systems," in *Proc. IEEE Int. Conf. Dependable Syst. Netw.*, Atlanta, GA, USA, 2014, pp. 1–2.

[23] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "OS Diversity for intrusion tolerance: Myth or reality?" in *Proc. IEEE/IFIP 41st Int. Conf. Dependable Syst. Netw.*, 2011, pp. 383–394.

[24] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "Analysis of operating system diversity for intrusion tolerance," *Softw.- Pract. Exp.*, vol. 44, no. 6, pp. 735–770, 2014.

[25] S. Frei, M. May, U. Fiedler, and B. Plattner, "Large-scale vulnerability analysis," in *Proc. SIGCOMM Workshop Large-Scale Attack Defense*, 2006, pp. 131–138.

[26] M. Shahzad, M. Zubair Shafiq, and A. Liu, "A large scale exploratory analysis of software vulnerability life cycles," in *Proc. 34th Int. Conf. Softw. Eng.*, 2012, pp. 771–781.

[27] L. Bilge and T. Dumitras, "Before we knew it: An empirical study of zero-day attacks in the real world," in *Proc. ACM Conf. Comput. Commun. Secur.*, Raleigh, NC, USA, 2012, pp. 833–844.

[28] B. Ladd, "The race between security professionals and adversaries," 2017. [Online]. Available: https://www.recordedfuture.com/vulnerability-disclosure-delay/

[29] A. Hahn and M. Govindarasu, "Cyber vulnerability disclosure policies for the smart grid," in *Proc. IEEE Power Energy Soc. Gen. Meeting*, San Diego, CA, USA, 2012, pp. 6675–6679.

[30] M. Cheung, "Market share analysis: Server operating systems, worldwide, 2015: Gartner report," 2016. [Online]. Available: https://www.gartner.com/doc/3326217/market-share-analysis-server-operating

[31] P. Tsai, "Server virtualization and OS trends," 2016. [Online]. Available: https://community.spiceworks.com/networking/articles/2462-server-virtualization-and-os-trends

[32] W3Techs, "Usage of operating systems for websites," 2017. [Online]. Available: https://w3techs.com/technologies/report/operating_system

[33] L. A. B. Sanguino and R. Uetz, "Software vulnerability analysis using CPE and CVE," *Cryptography Secur.*, arXiv:1705.05347. [Online]. Available: https://dblp.uni-trier.de/rec/bibtex/journals/corr/SanguinoU17

[34] M. Oiaga, "Recount: Windows still safest, tops Mac OS X, Linux and Sun Solaris. But are statistics a true measure of security?" 2007. [Online]. Available: http://news.softpedia.com/news/Recount-Windows-Still-Safest-Tops-Mac-OS-X-Linux-and-Sun-Solaris-57433.shtml

[35] J. Jones, "2006 client OS days of risk," 2007. [Online]. Available: https://blogs.microsoft.com/microsoftsecure/2007/06/18/2006-client-os-days-of-risk/

[36] H. Ghani, J. Luna, and N. Suri, "Quantitative assessment of software vulnerabilities based on economic-driven security metrics," in *Proc. Int. Conf. Risks Secur. Int. Syst.*, La Rochelle, France, 2013, pp. 1–8.

[37] A. Sajid, M. Ali Shah, M. Kamran, Q. Javaid, and S. Zhang, "An analysis on host vulnerability evaluation of modern operating systems," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 4, pp. 245–254, 2016.

[38] M. Kimura, "Software vulnerability: Definition, modelling, and practical evaluation for e-mail transfer software," *Int. J. Pressure Vessels Piping*, vol. 83, pp. 256–261, 2006.

[39] H. Okamura, M. Tokuzane, and T. Dohi, "Security evaluation for software system with vulnerability life cycle and user profiles (WDTS-RASD'2012)," in *Proc. Workshop Dependable Transp. Syst./Recent Adv. Softw. Dependability*, Niigata, Japan, 2012, pp. 39–44.

[40] Forum of Incident Response and Security Teams, "Common vulnerability scoring system, V3 development update," 2015. [Online]. Available: https://www.first.org/cvss

[41] National Vulnerability Database, "Vulnerability summary for CVE-2008-4609," 2008. [Online]. Available: https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-4609

[42] Cisco Systems, "TCP state manipulation denial of service vulnerabilities in multiple Cisco products," 2009. [Online]. Available: https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20090908-tcp24

[43] N. Paulauskas and E. Garsva, "Computer system attack classification," *Electron. Elect. Eng.*, vol. 2, no. 66, pp. 84–87, 2006.

[44] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Jan./Mar. 2004.

[45] P. Verissimo, N. Neves, and M. Correia, "The middleware architecture of MAFTIA: A blueprint," in *Proc. IEEE Inf. Survivability Workshop*, Boston, MA, USA, 2000, pp. 24–26.

[46] F. Majorczyk, E. Totel, and L. Me, "Experiments on COTS diversity as an intrusion detection and tolerance mechanism," in *Proc. Workshop Recent Adv. Intrusion-Tolerant Syst.*, Lisbon, Portugal, 2007, pp. 28–32.

[47] A. Valdes *et al.*, "An architecture for an adaptive intrusion-tolerant server," in *Security Protocols*. Berlin, Germany, Springer-Verlag, 2002, pp. 158–178.

[48] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, and B. Dutertre, "An adaptive intrusion-tolerant server architecture," in *Proc. 10th Int. Workshop Secur. Protocols*, 2002, pp. 17–19.

[49] A. Saidane, V. Nicomette, and Y. Deswarte, "The design of a generic intrusion-tolerant architecture for web servers," *IEEE Trans. Dependable Secure Comput.*, vol. 6, no. 1, pp. 45–58, Jan.–Mar. 2009.

[50] H. Moniz, N. Neves, M. Correia, and P. Verissimo, "RITAS: Services for randomized intrusion tolerance," *IEEE Trans. Dependable Secure Comput.*, vol. 8, no. 1, pp. 122–136, Jan./Feb. 2011.

[51] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Vienna, Austria, 2016, pp. 31–42.

[52] F. Osorio, "Using byzantine agreement in the design of IPS systems," in *Proc. IEEE Int. Perform., Comput. Commun. Conf.*, New Orleans, LA, USA, 2007, pp. 528–537.

[53] M. Correia, N. Neves, and P. Verissimo, "BFT-TO: Intrusion tolerance with less replicas," *Comput. J.*, vol. 56, no. 6, pp. 693–715, 2013.

[54] Y. Huang, D. Arsenault, and A. Sood, "Incorruptible system self-cleansing for intrusion tolerance," in *Proc. IEEE Int. Perform. Comput. Commun. Conf.*, Phoenix, AZ, USA, 2006, pp. 493–496.

[55] E. Totel, F. Majorczyk, and L. Me, "COTS diversity based intrusion detection and application to web servers," in *Recent Advances in Intrusion Detection* , A. Valdes and D. Zamboni, Eds. Berlin, Germany, Springer-Verlag, 2006, pp. 43–62.

[56] T. Distler, R. Kapitza, and H. Reiser, "State transfer for hypervisor-based proactive recovery of heterogeneous replicated," in *Proc. 'Sicherheit, Schutz und Zuverlässigkeit' Conf.*, Berlin, Germany, 2010, pp. 236–269.

[57] M. Castro, R. Rodrigues, and B. Liskov, "BASE: Using abstraction to improve fault tolerance," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 236–269, 2003.

[58] C. Pu, A. Black, C. Cowan, and J. Walpole, "A specialization toolkit to increase the diversity of operating systems," in *Proc. ICMAS Workshop Immunity-Based Syst.*, Nara, Japan, 1996, pp. 1–10.

[59] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin, "Separating agreement from execution for byzantine fault tolerant services," in *Proc. 19th ACM Symp. Operating Syst. Principles*, Bolton Landing, NY, USA, 2003, pp. 253–267.

[60] R. Schlichting and F. Schneider, "Fail-stop processors: an approach to designing fault-tolerant computing systems," *ACM Trans. Comput. Syst.*, vol. 1, no. 3, pp. 222–238, 1983.
[61] J. E. Just *et al.*, "Learning unknown attacks—A start," in *Recent Advances in Intrusion Detection*, A. Wespi, G. Vigna, and L. Deri, Eds. Berlin, Germany: Springer-Verlag, 2002, pp. 158–176.
[62] A. Bessani *et al.*, "SCFS: A shared cloud-backed file system," in *Proc. USENIX Annu. Tech. Conf.*, Philadelphia, PA, USA, 2014, pp. 169–180.

Anatoliy Gorbenko received the M.Eng. degree in computer engineering and the Ph.D. degree in computer science from National Aerospace University, Kharkiv, Ukraine, in 2000 and 2005, respectively, and the D.Sc. habilitation degree and Full Professorship in computer science with the Department of Computer Systems and Networks, National Aerospace University, in 2012.

From 2014 to 2016, he was the Dean of the Aircraft Radio-Technical Faculty and the Leader of the Service-Oriented Systems Dependability research group. In 2017, he joined the School of Computing, Creative Technologies & Engineering, Leeds Beckett University, Leeds, U.K. He is also a Visiting Professor with the Department of Computer Systems and Networks, National Aerospace University. His expertise and research interests include dependability and performance of distributed systems, SOA and clouds, and SW vulnerability and intrusion tolerance.

Alexander Romanovsky received the M.Sc. degree in applied mathematics from Moscow State University, Moscow, Russia, in 1976, and the Ph.D. degree in computer science from St. Petersburg State Technical University, St. Petersburg, Russia, in 1988.

From 1993 to 1994, he was a Postdoctoral Fellow with the Department of Computing Science, University of Newcastle upon Tyne, Newcastle upon Tyne, U.K. He is currently a Professor in Computing Science with the School of Computing, Newcastle University, Newcastle upon Tyne, and the Investigator of the PRiME programme and the STRATA platform EPSRC/UK grants. From 1984 to 1996, he was with St. Petersburg State Technical University, doing research and teaching. His main research interests include system dependability, fault tolerance, software architectures, exception handling, error recovery, and system verification for safety.

Prof. Romanovsky is a member of the editorial boards of *Computer Journal* and the IEEE TRANSACTIONS ON RELIABILITY.

Olga Tarasyuk received the B.S. and M.S. degrees in computer engineering and the Ph.D. degree in computing science from National Aerospace University, Kharkiv, Ukraine, in 2000, 2001, and 2004, respectively.

Since 2005, she has been an Associate Professor in Computing Science with the Department of Computer Systems and Networks, National Aerospace University. Her main expertise include software quality and reliability, data analytics, development of dependable big-data, and cloud computing solutions focusing on tradeoffs between consistency, availability, performance, and partition tolerance.

Oleksandr Biloborodov received the B.S. and M.S. degrees in computing science in 2011 and 2013, respectively, from National Aerospace University, Kharkiv, Ukraine, where he is currently working toward the Ph.D. degree in computing science.

Since 2014, he has been a Senior Software Developer with Plarium LLC, Kharkiv, Ukraine. His main expertise include quality of software and software engineering process, intrusion tolerance, and vulnerability analysis.