CERIAS Tech Report 2005-05

A TREND ANALYSIS OF VULNERABILITIES

by Rajeev Gopalakrishna and Eugene H. Spafford

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

A Trend Analysis of Vulnerabilities

Rajeev Gopalakrishna and Eugene H. Spafford
CERIAS, Purdue University, USA
1315 Recitation Building
656 Oval Drive, West Lafayette, IN 47907-2086
Email: {rgk,spaf}@cerias.purdue.edu

Abstract

Software vulnerabilities exist and will continue to do so. Every week, a new vulnerability gains popular attention, is discussed at length in mailing lists, and hopefully gets patched by the vendor before exploits and attack tools start appearing. But there is little evidence that we are learning from our mistakes. Sharing of vulnerability information through public databases has been possible for quite sometime now. If it is not lack of information, what is it that is preventing us from learning from our past? Are there any lessons to be learned at all? A good start towards answering such questions would be to analyze vulnerabilities in widely deployed, critical but buggy software artifacts. In this paper, we look at vulnerabilities in five such software artifacts and examine two of their attributes. Among other statistics, our analysis suggests that the discovery of a vulnerability in a software artifact may influence the discovery of more vulnerabilities of the same type in that artifact. Thus, there may be some learning occurring, but it is by the penetration community rather than the software engineers. This paper argues that measuring vulnerability occurrences may have predictive value and that this concept of retrospective metric is an interesting approach to expressing assurance.

Keywords: Vulnerability Analysis, Security Metrics, Assurance

1 Introduction

The number of software vulnerabilities reported is increasing every year. The number of new vulnerabilities that made their way into the ICAT metabase [3] was 859 in 1999, 990 in 2000, 1506 in 2001, and 1307 in 2002. This brings up some interesting questions — is there a pattern in the discovery of vulnerabilities? Are certain types of vulnerabilities more common than the others? If so, what combination of vulnerability prevention mechanisms would give the maximum coverage?

A breach or an intrusion process in the context of security can be considered analogous to a failure in the context of reliability [17]. There have been studies based on empirical data that suggest that the times between security breaches might be exponentially distributed [14]. This implies that well-studied and extensively used probabilistic methods for reliability modeling such as Markov models could be used on security breaches. While this might lead us to a useful metric such as the *mean time to breach*, the analysis of the underlying vulnerabilities might lend itself to a similar metric, namely the *mean time to next vulnerability*. This could also give rise to other useful indicators such as the mean number of vulnerabilities that one could expect in the installed software products, and the mean number of patches that would be released for a particular software product during a certain period. Such projected estimates could be helpful in allocating resources for the expected situation. It could also serve as a metric for rating software vendors. The vendors themselves could use such metrics for assessing and improving the security of their software products, thus providing greater assurance to the users.

Vulnerabilities come in many types: buffer overflows, race conditions, environmental errors, and design errors among others. There are different techniques to detect the presence of different vulnerability types. Because prioritizing efforts is often crucial in any resource-constrained and time-critical task such as the detection and prevention of vulnerabilities, a classification of vulnerabilities in a software artifact into the different types might assist in directing the time and resources to the more critical ones. For example, if 60% of the vulnerabilities in a software artifact are design errors and only 20% are coding errors, it would perhaps make more sense to review the design decisions for the software artifact before analyzing its code.

With the aim of gaining insight into such issues, we examined five widely deployed software artifacts that have been in widespread use for at least five years and that have had significant vulnerabilities. The vulnerabilities in these

software artifacts have been listed among the 20 most critical Internet security vulnerabilities by the SANS institute [6]. The wide use of the software artifacts and the criticality of their vulnerabilities make them good candidates for studying trends.

We aggregated information on the vulnerabilities from publicly available sources such as Bugtraq [1], ICAT, and CVE [2] and then analyzed their pattern of occurrence and their characteristics. The analysis provides evidence for the influencing nature of vulnerability occurrences and hence warrants further research in the area of predictive vulnerability analysis whose results can be used as a "retrospective" metric to reflect on past vulnerabilities and direct future efforts in developing software products with fewer vulnerabilities.

2 Scope

In this section, we describe the scope and limitations of this study and of the resulting observations. First, the vulnerabilities that we analyze are a subset of known software vulnerabilities. One could postulate that the set of known vulnerabilities is not representative of all the vulnerabilities present in the system because there might be many (possibly unique) that have not been detected so far. One could also hypothesize that not all known vulnerabilities are reported or made public. But considering that the recent trend has been one of full disclosure, we can assume that most vulnerabilities that are exploited or discovered are reported (or made public). If this is not the case, then the suite of security tools including intrusion detection systems, integrity checkers, and forensic analyzers might well detect those exploits and reveal the underlying vulnerabilities. Hence we conclude it is reasonable to assume that analyzing the reported vulnerabilities is sufficient.

Second, we look at vulnerabilities in only five software artifacts drawn from two system classes (Unix and Windows). These software artifacts are high profile in nature and have a significant presence in standard computing systems. They are presumably written with extra care because of their criticality. And they have also been shown to have serious vulnerabilities in the past.

3 Candidate Software Artifacts

In this section, we elaborate on the criteria used to narrow our focus to the five selected software artifacts. The characteristics we desired in the software artifacts are:

- *The software artifact should not be an operating system*

  Operating systems are complex software artifacts supporting numerous functions. They usually have a wider developer base than application software products and are probably debugged more extensively because of their criticality. They have a longer life cycle also. In effect, they would introduce many different variables into our analysis. Therefore, we confined ourselves to applications for the purposes of this study.

- *The software artifact should have been deployed for at least two years*

  Two years time should be good enough for a software product to draw attention and gain acceptance in the user community. This would also give sufficient time for vulnerabilities to begin to be found and reported. These two assumptions depend upon the type of the software product. If the software product targets a limited consumer base, it might not get the exposure that we assume is necessary for our study.

- *The software artifact should be widely deployed*

  The software product to be considered must have enough scope and market penetration to make it an attractive target for users to examine to find vulnerabilities. This is what we refer to as the "sufficient-eyeballs phenomenon" much like the "many-eyeballs phenomenon" associated with open source software [20].

- *The software artifact should have had significant vulnerabilities*

  Here we use the term "significant" in terms of both the number of vulnerabilities as well as the nature of vulnerabilities. A software product that has a couple of vulnerabilities that are exploitable only by local users (and not remotely) is not "significant" for our purposes. It would not be a good candidate to predict trends.

In October of 2001, the SANS Institute published a list of 20 "most critical" vulnerability types organized into three categories: vulnerabilities that affect all systems, those that affect Windows systems, and those that affect Unix systems. Attack tools are widely available for these vulnerabilities, making them accountable for a significant number of successful attacks. This SANS/FBI top 20 list was also put together by a team of security experts drawn from places including federal agencies, security software

vendors, consulting firms, academia, CERT/CC, and the SANS institute. These attributes of the list make it a primary source for choosing software artifacts for our research. The candidate software artifacts that were reduced from this list are given below with a brief description of their functions:

1. Microsoft Internet Information Server (IIS)

   IIS is the web server software product from Microsoft. According to the Netcraft web server survey [5], it had about 25% market share among the web servers in August, 2002. It has had a variety of vulnerabilities ranging from buffer overflows to configuration errors.

2. Berkeley Internet Name Domain (BIND)

   BIND is the most widely used implementation of the Domain Name System (DNS) protocols whose components include a DNS server (`named`), a DNS resolver library, and some support tools. Because of its importance in the functioning of the Internet (it translates between domain names and IP addresses), it has been a common target for attack. The SANS list refers to a mid-1999 survey that showed that as many as 50% of all the DNS servers on the Internet were running vulnerable versions of BIND.

3. Line Printer Daemon (Lpd)

   Lpd is the line printer daemon found in most Unix systems. It provides services to allow users to interact with the local printer.

4. Sendmail

   Sendmail is the most widely used mail-transfer software on the Internet. It has seen several vulnerabilities since the one exploited by the infamous Morris worm [19].

5. Remote Procedure Call (RPC)

   RPC services enable programs on one computer to execute code on another computer across a network. The very nature of their functionality makes them an attractive attack target.

The above software artifacts are available in different versions for different operating systems and sometimes in different forms (RPCs). For every software artifact, we consider vulnerabilities across all these boundaries. This implies that the vulnerabilities considered for a particular software artifact might not necessarily be in the source code for each platform (though code sharing is common across Unix-like operating systems).

4 Vulnerability Information Sources

The foundation of this study rests on accurate vulnerability information. Hence we refer to three popular and widely used sources of vulnerability information namely CVE, Bugtraq, and ICAT. Below, we give their brief descriptions:

- *Common Vulnerabilities and Exposures (CVE)* is a list of standardized names for publicly known vulnerabilities and other security exposures. It facilitates sharing of information across different vulnerability databases. In our study, CVE names have been used to compare vulnerability information between ICAT and Bugtraq.
- *Bugtraq* is a full disclosure, moderated mailing list for the detailed discussion and announcement of computer security vulnerabilities. Started in 1993, it has grown into a widely used mailing list with over thirty thousand subscribers. However, the submitted information is neither verified nor validated.
- *ICAT* is a searchable index of computer security vulnerability information. It enables searching for vulnerability information at a fine granularity based on different attributes such as vendor, product, product version, exploit range, vulnerability consequence, vulnerability type, and exposed component type. Besides providing a summary of the vulnerability and describing its attributes, it references other vulnerability databases and vendor sites that provide related information, thus serving as a metabase.

5 Vulnerability Attributes

A vulnerability has certain characteristics based on the nature of the vulnerability and the impact that it has on a system. The time of its discovery (or reporting) is a feature external to the vulnerability itself, although useful for our study. In this section, we describe the methodology employed in obtaining the published date and type of a vulnerability from the two sources (ICAT and Bugtraq). For vulnerabilities present in both the databases, we employ a simple heuristic to obtain what we believe to be the more correct data. And for vulnerabilities present in only one database, there is no choice but to use the available information.

5.1 Date Published

As previously mentioned, the trend in the appearance of vulnerabilities is of interest to us. Though a vulnerabil-

ity might have been discovered well before its presence is disclosed to (discussed by, shared with, or detected by) a public audience such as Bugtraq, it is the best approximation that one can hope to achieve while using real-world data. It is because of this uncertainty that we restrict ourselves to only the month and year of publishing without considering the exact day when the vulnerability was supposed to have been reported.

The ICAT documentation mentions that the "published before" attribute for older vulnerabilities listed on ICAT is the earliest dating among publicly available sources. But for vulnerabilities reported in the year 2001 and beyond, they use the date that the vulnerability was added to their list. Bugtraq has a longer history than ICAT, and also has "published before" information for the vulnerabilities listed in it. So as a general rule, the earlier of the two dates from ICAT and Bugtraq is used for the purposes of the study.

5.2 Vulnerability Type

Multiple classification schemes exist for vulnerabilities as described by Aslam, Krsul, and Spafford [8, 16]. Both ICAT and Bugtraq have their own taxonomies for vulnerabilities and provide the type information for every vulnerability. We chose ICAT's classification scheme over Bugtraq's for two reasons. ICAT has documentation explaining its classification and Bugtraq does not. Second, ICAT is from the National Institute of Standards and Technology (NIST) [4], which can be viewed as a more considered and reputable source. We use Bugtraq's vulnerability type information only in cases where the vulnerability was not listed in ICAT, or when ICAT classified the vulnerability type as unknown while Bugtraq specified a particular type.

According to ICAT's classification scheme, a vulnerability can be of the following types: *input validation error, access validation error, exceptional condition handling error, environmental error, configuration error, race condition error, design error,* or *other (unknown)*. An input validation error can be a *boundary condition error*, *buffer overflow error*, or of some other type. This is not a true classification scheme because occasionally a vulnerability that exhibits multiple characteristics is assigned more than one type. The ICAT documentation contains the working definitions of these different vulnerability types.

In this study, Aslam et al.'s classification scheme [8] is used to visualize a higher level classification of the vulnerability types[1]. The input validation, access validation, and exceptional condition handling errors are grouped under the *conditional validation* category. Race condition errors (atomicity or synchronization errors) and the rest map onto themselves. The next level of classification combines condition validation and synchronization errors into *coding errors*. Configuration and environment errors are grouped under *emergent errors*. We deviate from this classification scheme in that we retain design error in the highest level without grouping it under coding errors. This stems from the fact that a design error need not be a coding error. An example of a design error that is not a coding error is the use of a weak encryption algorithm in a software product that creates a window of opportunity for someone to break in. In this case, it was the design decision to use a weak algorithm that created the vulnerability.

[1] See Table 1

6 Analysis

This section deals with numbers and graphs. We present and analyze the yearly distribution of vulnerabilities, the differential time intervals between successive vulnerabilities, the type groups into which they fall, and their statistical dependency. Vulnerability information was collected in June of 2002.

6.1 Trend

The yearly trend in the appearance of vulnerabilities is represented in figures 1-5. In the absence of any striking patterns, we make a few observations:

1. Among the five software artifacts, the highest number of vulnerabilities reported in a year is 28 for Microsoft IIS in 1999.

2. With the exception of Lpd, the year 1999 seems to have had the maximum reports of vulnerabilities for the software artifacts we studied. One possible explanation for this high incidence of vulnerability reports in 1999 could be the heightened interest in Y2K related software errors in that year.

6.2 Differential Time

The differential time [2] between successive vulnerability reports for a software artifact gives a notion of their inter-arrival time (See figures 6-10). A general observation across all the five software artifacts is that the differential

[2] The differential time for vulnerability **n** is the number of months between the discoveries (reports) of vulnerabilities **n-1** and **n**. We ignored the first vulnerability for differential analysis (used a value of zero) because the initial release dates for the software artifacts were not determined.

times generally tend to be longer in the first few years after the release of the software. One possible explanation for this trend can be that a certain period of time is needed for a "critical-mass" of users to start using the software product and find bugs in it (what we refer to as the "sufficient-eyeballs" phenomenon). From then on, vulnerabilities are uncovered at a fairly constant rate. This reasoning ignores the facts that the accounted vulnerabilities are for software artifacts across different operating systems, and that the different versions and patches might modify the software code considerably. Some other observations that might be of interest are as follows:

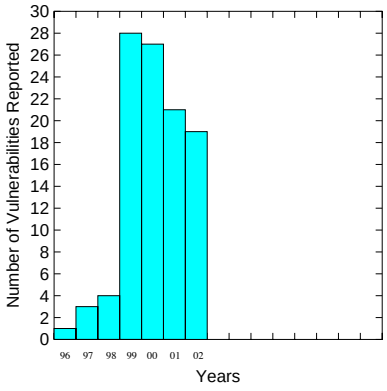

Figure 1: Yearly Trend of IIS Vulnerabilities

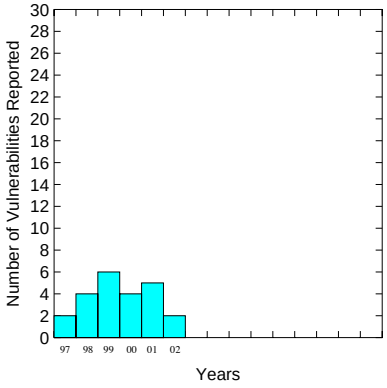

Figure 2: Yearly Trend of BIND Vulnerabilities

1. The longest gap between successive reports of vulnerabilities is 49 months (just over four years) between the third and fourth vulnerabilities in Sendmail.

2. The differential times for all the software artifacts has peaked at or before the seventh vulnerability report.

3. The maximum number of vulnerabilities reported in a single month (longest streak of zero differential times) is 13 for Microsoft IIS, which is almost one every other day.

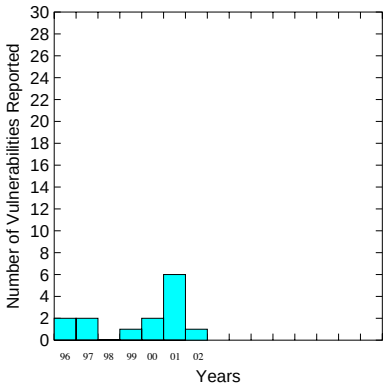

Figure 3: Yearly Trend of Lpd Vulnerabilities

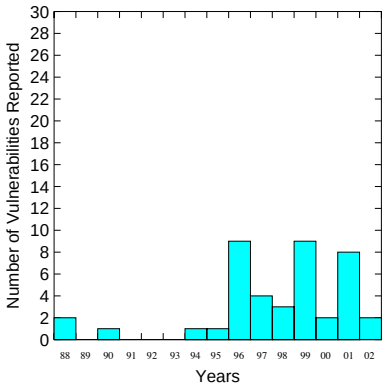

Figure 4: Yearly Trend of Sendmail Vulnerabilities

6.3 Type

Figures 11-15 present the percentages of the eight vulnerability categories for the five software artifacts. These charts give a better indication of the extent to which the different vulnerability types have appeared in these software artifacts. We make the following observations:

1. Input validation errors clearly dominate in all the software artifacts, ranging from 43% to 70% approximately.

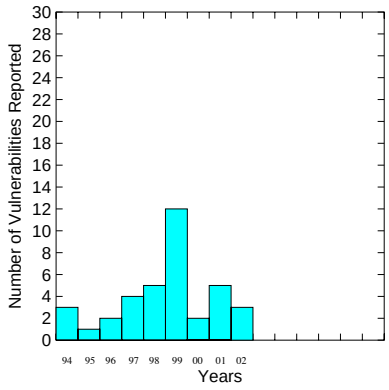

Figure 5: Yearly Trend of RPC Vulnerabilities

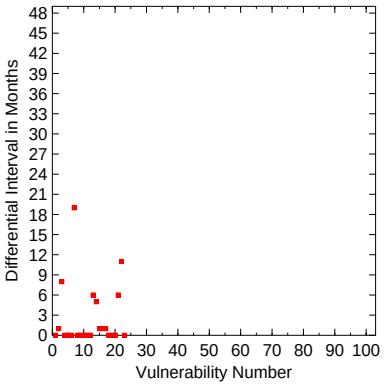

Figure 7: Differential Times for BIND Vulnerabilities

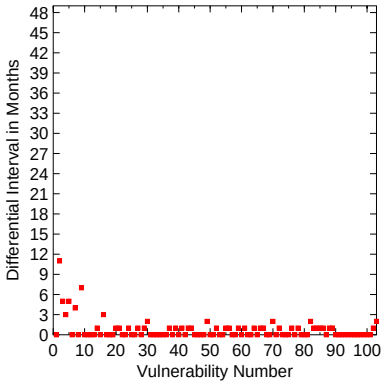

Figure 6: Differential Times for IIS Vulnerabilities

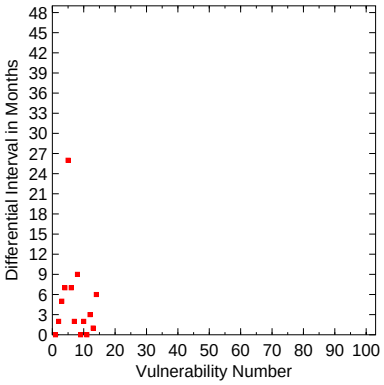

Figure 8: Differential Times for Lpd Vulnerabilities

2. Design errors and access validation errors come second in a majority of the software artifacts.

3. Among RPC vulnerabilities, almost 22% of the vulnerabilities are of unknown type. This might be attributed to one of the two reasons. The vulnerabilities might not clearly fall into the categories chosen, in which case, it might be an indication that a better vulnerability classification scheme is necessary (as suggested in the ICAT documentation). Or else, given the resource constraints, it might have been difficult for the ICAT team to determine the right category, leading them to classify it as unknown for the time being.

6.4 Aggregate Analysis

In this section, we analyze vulnerabilities across all the five software artifacts and make observations about the vulnerability distribution. Table 1 shows the distribution of vulnerability categories across all the software artifacts[3]. We make the following observations:

1. Similar to the observations made in section 6.3, input validation errors dominate the vulnerabilities followed by design and access validation errors.

2. Buffer Overflows constitute nearly 20% (one-fifth) of all the vulnerabilities.

3. Condition validation errors account for nearly 70% of all the vulnerabilities.

4. Coding errors account for almost 72% of all the vulnerabilities.

5. Approximately, about 95% of all vulnerabilities originate during software development. (Only the environment errors are attributed to the user. From the

[3]Percentages add up to more than 100% because some vulnerabilities fall in more than one category.

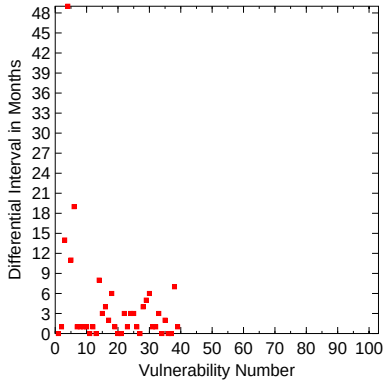

Figure 9: Differential Times for Sendmail Vulnerabilities

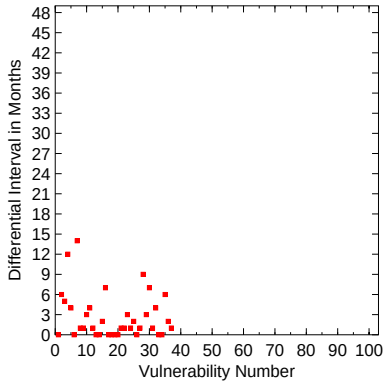

Figure 10: Differential Times for RPC Vulnerabilities

definition of configuration errors, these arise when the developer ships software products with weak configuration and therefore are attributed to the developer)

ICAT has similar statistics showing the percentages of different vulnerability categories across all CVE and CVE candidate entries published in ICAT on a yearly basis (from 1999 to 2002). In Table 2, we compare the percentages for our five software artifacts against those from ICAT statistics (averaged over the four years). We see that the percentages are comparable for most categories except input validation, design, and unknown categories. While the design errors being comparatively lesser in our case may be a positive sign, this reinforces our observation that the five software artifacts with critical vulnerabilities are overly susceptible to input validation errors. It also suggests that critical software products have a greater percentage of vulnerabilities in the unknown category (7.4% compared to 0.5%).

If we analyze the 216 aggregate vulnerabilities, we find that the number of times two consecutively reported vulnerabilities are of the same type is 68 [4] (calculated separately for each software and summed up). This means that almost one-third (31.5%) of the times when two successive vulnerabilities are considered in a software artifact, they are of the same type. On a per software product basis, the numbers are 12 (30.8%) for Sendmail, 30 (29.1%) for IIS, 11 (47.8%) for BIND, 5 (35.7%) for Lpd, and 10 (27%) for RPC. So on an individual basis, the fraction of successive vulnerabilities being the same type varies roughly from one-fourth (for RPC) to one-half (for BIND). These numbers indicate that the mean for the five software artifacts of almost one-third is reasonably influenced by all the software items in the absence of any significant outliers.

Consider the following situation. A user detects a vulnerability in a software product. The user may be any person not associated with the software vendor such as a casual user, a hacker, or a cracker (one with malicious intent). Depending on the category to which the user belongs, the details of the discovered vulnerability propagate to a certain population at a certain speed. A natural outcome of learning is to apply the lessons learned to a similar domain. In this case, it could mean trying to find similar vulnerabilities in other parts of the same software product. From the users' perspective, this behavior could result because of the available experience (either their own or of others) in discovering vulnerabilities of that particular kind. From the vendors' perspective, the people and processes responsible for the vulnerable software product could be particularly incompetent in avoiding this type of vulnerabilities (buffer overflows for example) thus contributing to the existence of this phenomenon. This reasoning can be further extended to systems harboring the vulnerabilities, where the policies and mechanisms in place are particularly incapable of detecting and resolving vulnerabilities of this kind. This phenomenon is of interest while considering the independence of vulnerability reports because this observation tends to suggest that the report of a particular vulnerability might influence the discovery of similar vulnerabilities. If the vulnerability reports are not independent then the times between such reports are not exponentially distributed. This would rule out the application of Markovian models and probabilistic metrics based on it. And if it can be argued that every vulnerability can be mapped to a reasonably disclosed attack tool and that a certain constant time would be sufficient to develop an attack tool after the vul-

[4] In cases where a vulnerability fell into more than one category, we considered both types for this calculation.

nerability is known, then the times to breach (attack) also would not be exponentially distributed. This would potentially affect some of the research in the area of intrusion process modeling and quantitative security metrics, and might warrant revisiting some of the assumptions made in those studies.

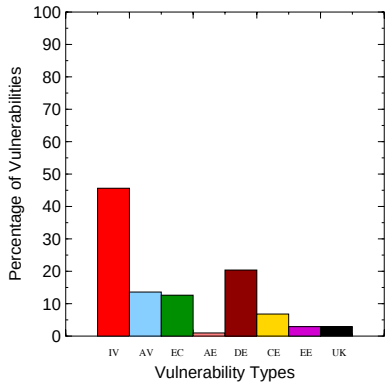

Figure 11: Distribution for IIS Vulnerabilities

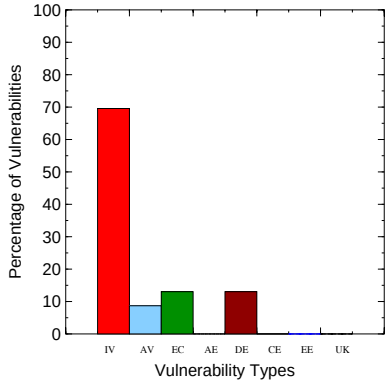

Figure 12: Distribution for BIND Vulnerabilities

However, the percentages calculated could be skewed because of the fact that nearly 50% of the aggregated vulnerabilities belong to the input validation category. Intuitively, consecutive pairs of this type would appear nearly 25% of the time in a random draw. When we randomized the sequence of vulnerabilities in the individual softwares and performed a similar calculation, the numbers we get are 7 (17.9%) for Sendmail, 32.4 (31.5%) for IIS, 10.2 (44.3%) for BIND, 5.4 (38.6%) for Lpd, 8.2 (22.2%) for RPC, and 63.2 (29.3%) for the aggregate (all numbers averaged over five runs). Besides Sendmail, the difference between the actual numbers and the corresponding numbers for the randomized sequences is not too large. This

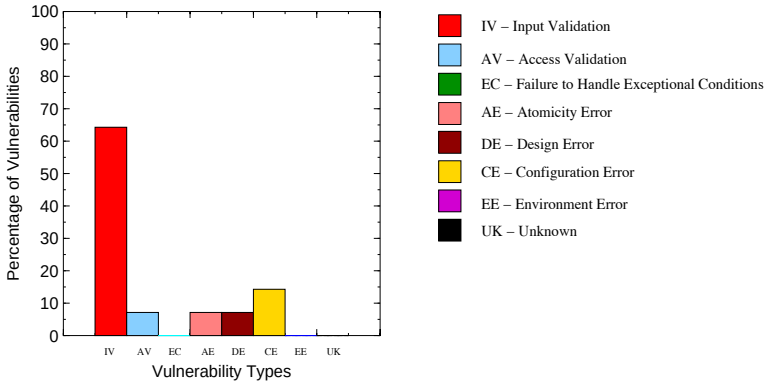

Figure 13: Distribution for Lpd Vulnerabilities

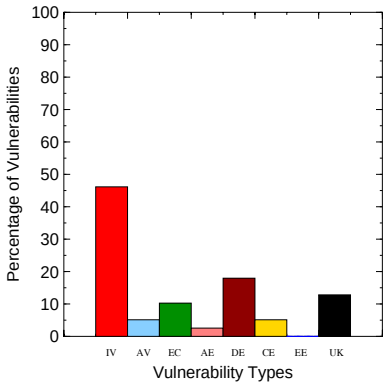

Figure 14: Distribution for Sendmail Vulnerabilities

prompted us to further investigate this phenomenon using statistically accepted procedures, the details of which are presented in the next subsection.

6.5 Test for Statistical Independence

A statistical procedure generally used for evaluating the statistical independence of a sequence of observations is the *run test* [9]. Table 3 summarizes the results of the run test analysis (performed at a confidence level of 95%) of the different vulnerabilities categories for the five software artifacts. Software items that did not have any vulnerabilities in a particular category have a "-" entry in the table. We make the following observations:

1. Out of the 32 non-null entries in the table, vulnerability reports in 24 categories are statistically dependent. This supports our conjecture that vulnerability reports may influence the discovery of similar

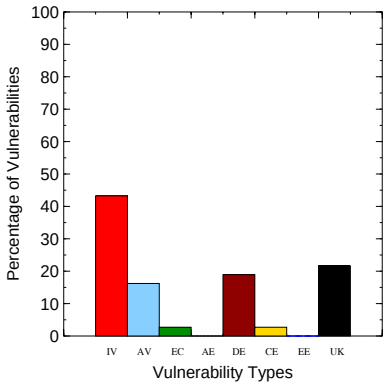

Figure 15: Distribution for RPC Vulnerabilities

vulnerabilities in a majority of the categories for the five software artifacts.

2. All vulnerability categories of Sendmail are dependent. We speculate that this could be attributed to the fact that Sendmail was mostly written by one person. Therefore, the errors could be highly repetitive in nature.

3. Except for Sendmail, the input validation category of vulnerabilities is independent for the other four software artifacts. One possible explanation for this is the fact that input validation category is a broad category that includes vulnerabilities such as buffer overflows, boundary condition errors, and format string vulnerabilities.

4. The access validation error, failure to handle exceptional conditions, and atomicity error categories are dependent for all the software artifacts.

7 A Retrospective Metric

> *"If you can measure that of which you speak and express it in numbers, you know something about your subject; but if you cannot measure it, your knowledge is of a very meager and unsatisfactory kind"*
> Lord Kelvin

If Lord Kelvin is correct, then our knowledge of issues such as the security of a system, effectiveness of a security product, and susceptibility of a software product to vulnerabilities (among other things) is of a meager kind. There has certainly been a reasonable amount of interest in the area of security metrics [7, 10, 13, 14, 15, 17, 18, 21]. But, it is still an immature discipline. While we do not attempt to measure any snapshot of the security state of a system or product, the observations made about vulnerabilities and types might serve as a metric to prioritize security objectives and also aid other research on security metrics.

An observation we made was that coding errors constitute nearly 72% of the analyzed vulnerabilities. An interesting aspect to study would be to determine a combination of coding and testing practices that would have prevented the maximum number of such vulnerabilities. Along similar lines, it would be worthwhile to compare the effectiveness of the various buffer overflow prevention mechanisms (such as non-executable stacks and introduction of "canaries" [12]) in thwarting the buffer overflows that constitute nearly 20% of the analyzed vulnerabilities. These are examples of what we call the *retrospective metric*. This would be a metric in the true sense because it would for example, measure the extent of effectiveness of the vulnerability prevention mechanisms, albeit for known vulnerabilities. If the preventive measures offer protection against certain genres of vulnerabilities and if we can extrapolate the generic (category-wise) distribution of vulnerabilities, then the retrospective metric could be extended to unseen vulnerabilities also. Such a metric could be used to quantify a class of vulnerabilities that constitutes a certain percentage of all vulnerabilities reported for that software product. An example would be a metric stating that the use of a combination of techniques A, B, and C during the software development lifecycle can increase the assurance in the resulting software product by 10% (by avoiding a class of vulnerabilities that constitutes 10% of all vulnerabilities reported for that software product).

For a retrospective metric to be useful in the future against unknown vulnerabilities, we should be able to forecast the future by looking at the past vulnerabilities. From the observations we have made regarding the statistical independence of vulnerabilities, it appears that any such forecasting or modeling would have to be done on an individual basis (category-wise). If the vulnerability reports in a category are dependent then we can use time-series analysis. If they are independent then we can use Markovian models to represent them. In either case, we might be able to get a probabilistic estimate of the number and the times of arrival of future vulnerabilities. A desirable goal would be to predict trends in the discovery of vulnerabilities similar to the work of Browne et al. [11], which looks at statistical trends in reported intrusion incidents involving exploited vulnerabilities. However, in this paper, we do not attempt such predictions because of one main

Vulnerability Category	Vulnerability Sub-Category	Vulnerability Type	Percentage
Coding	Condition Validation	Input Validation Error	49.07
		(Buffer Overflow)	(20.37)
		(Boundary Condition)	(6.02)
Coding	Condition Validation	Access Validation Error	11.57
Coding	Condition Validation	Failure to Handle Exceptional Conditions	9.72
Coding	Synchronization	Atomicity Error	1.39
Design	Design	Design Error	18.06
Emergent	Configuration	Configuration Error	5.56
Emergent	Environment	Environment Error	1.39
Unknown			7.41

Table 1: Distribution of Vulnerability Categories across the Five Software Artifacts.

Vulnerability Type	Our Statistics (percentage)	ICAT Statistics (percentage)
Input Validation Error	49.07	41
(Buffer Overflow)	(20.37)	(19.5)
(Boundary Condition)	(6.02)	(4.25)
Access Validation Error	11.57	11.75
Failure to Handle Exceptional Conditions	9.72	9
Atomicity Error	1.39	2.25
Design Error	18.06	24
Configuration Error	5.56	7.25
Environment Error	1.39	2
Unknown	7.41	0.5

Table 2: Comparison of Vulnerability Category Distribution between the Aggregate for our Five Software Artifacts and the ICAT Statistics for all of its CVE Entries.

Vulnerability Type	IIS	BIND	Lpd	Sendmail	RPC
Input Validation Error	Independent	Independent	Independent	Dependent	Independent
(Buffer Overflow)	Dependent	Independent	Independent	Dependent	Dependent
Access Validation Error	Dependent	Dependent	Dependent	Dependent	Dependent
Failure to Handle Exceptional Conditions	Dependent	Dependent	-	Dependent	Dependent
Atomicity Error	Dependent	-	Dependent	Dependent	-
Design Error	Dependent	Dependent	Dependent	Dependent	Independent
Configuration Error	Dependent	-	Independent	Dependent	Dependent
Environmental Error	Dependent	-	-	-	-

Table 3: Run Test Analysis of the Statistical Independence of Different Vulnerability Categories for the Five Software Artifacts.

reason — the data gleaned from the public databases may not yet be sufficient and accurate enough to make such claims with reasonable confidence.

8 Conclusions

We have analyzed software vulnerabilities in five critical software artifacts using information from public databases. The goal was to observe trends in the data and determine if they suggest something new or support something that we suspect. The trends noted in the temporal and spatial distributions of vulnerabilities might mean more (or less) than what we have inferred. We cannot claim that the results of this study will be applicable to every other software artifact or even a majority of them. But we can conjecture that for software products of similar nature and in similar environments (such as absence of laws prohibiting disclosure of vulnerabilities and lack of liability on vendors for vulnerabilities discovered in their software products) there will be a tendency to behave like the others in their group.

Our speculation that a vulnerability report might influence future discovery of similar vulnerabilities seems to be supported by the statistical dependency tests. The proof of existence of this phenomenon can have significant ramifications. This along with the other observations suggest that the concept of a retrospective metric warrants further study. A more populated, focussed, detailed, and accurate vulnerability information repository might generate more trustworthy observations and enable one to make predictions with a reasonable degree of confidence.

References

[1] Bugtraq, http://www.securityfocus.com/bid/.

[2] Common Vulnerabilities and Exposures (CVE), http://www.cve.mitre.org/.

[3] ICAT Metabase, http://icat.nist.gov/.

[4] National Institute of Standards and Technology. http://www.nist.gov/.

[5] Netcraft Web Server Survey. http://www.netcraft.com/survey/.

[6] The Twenty Most Critical Internet Security Vulnerabilities. The SANS Institute, http://www.sans.org/top20.htm/.

[7] Workshop on Information-Security-System Rating and Ranking. Williamsburg, Virginia, May 2001.

[8] T. Aslam, I. Krsul, and E. H. Spafford. Use of a taxonomy of security faults. In *Proc. 19th NIST-NCSC National Information Systems Security Conference*, pages 551–560, October 1996.

[9] Julius S. Bendat and Allan G. Piersol. *Random Data - Analysis and Mesaurement Procedures*. John Wiley and Sons, second edition, 1986.

[10] S. Brocklehurst, T. Olovsson, B. Littlewood, and E. Jonsson. On measurement of operational security. Technical Report PDCS No. 160, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden and Centre for Software Reliability, City University, London, April 1994.

[11] H. K. Browne, W. A. Arbaugh, J. McHugh, and W. L. Fithen. A trend analysis of exploitations. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 214–229, May 2001.

[12] Crispan Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, Qian Zhang, and Heather Hinton. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *7th USENIX Security Conference*, pages 63–78, San Antonio, Texas, january 1998.

[13] M. Dacier, Y. Deswarte, and M. Kaniche. Quantitative assessment of operational security: Models and tools. Technical Report LAAS Report 96493, May 1996.

[14] Erland Jonsson and Tomas Olovsson. A quantitative model of the security intrusion process based on attacker behavior. *IEEE Transactions on Software Engineering*, 23(4), April 1997.

[15] Stuart Katzke. Security metrics - what are they? CSSPAB Workshop on Approaches to Measuring Security, June 2000 .

[16] I. Krsul. *Software Vulnerability Analysis*. PhD thesis, Purdue University, Department of Computer Sciences, West Lafayette, Indiana, 1998.

[17] B. Littlewood, S. Brocklehurst, N. Fenton, P. Mellor, S. Page, D. Wright, J. Dobson, J. McDermid, and D. Gollmann. Towards operational measures of computer security. *Computer Security*, 2:211–229, 1993.

[18] Gregg Schudel and Bradley J. Wood. Adversary work factor as a metric for information assurance. In *Proceedings of the New Security Paradigms Workshop, Cork, Ireland*, September 2000.

[19] E.H. Spafford. The Internet worm: Crisis and aftermath. *Communications of the ACM*, 32(6):678–688, June 1989.

[20] John Viega and Gary McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison Wesley Professional, 1st edition, 2001.

[21] C. Wang and W. Wulf. A framework for security measurement. In *National Information Systems Security Conference, Baltimore, MD, pp. 522-533*, October 1997.