Software Security Growth Modeling: Examining Vulnerabilities with Reliability Growth Models

Andy Ozment*

Computer Security Group
Computer Laboratory, University of Cambridge

Abstract. The software engineering tools historically used to examine faults can also be used to examine vulnerabilities and the rate at which they are discovered. I discuss the challenges of the collection process and compare two sets of vulnerability characterization criteria. I collected fifty-four months of vulnerability data for OpenBSD 2.2 and applied seven reliability growth models to the two data sets. These models only passed applicability tests for the data set that omits dependent data points. Musa's Logarithmic model has the best one-step-ahead predictive accuracy of the three acceptably accurate models for that data set. It estimated that fifty-four months after OpenBSD 2.2's release, the mean time to vulnerability discovery for OpenBSD 2.2 was 42.5 days and that 58.4% of the vulnerabilities it contains had been found. However, a trend analysis cannot rule out the possibility that there is no trend at all in the rate of vulnerability detection, and this result casts doubts on the accuracy of the reliability growth models. The lack of a clear decreasing trend in that analysis highlights one of the challenges of using reliability growth models on vulnerability data: it may be a true reflection of the system or it may be caused by the changes over time in the effort invested in vulnerability detection.

1 Introduction

Most commercial software suffers from significant design and implementation security vulnerabilities. This lack of security can be traced to two primary factors: complexity and motivation. Software developers push to create ever more complex products and work constantly on the boundary of manageable complexity. However, even taking this difficulty into account, most software contains security flaws that its creators were readily capable of preventing. The second cause of software insecurity is motivation: although vendors are capable of creating more secure software, the economics of the software industry provide them with little incentive. Consumers generally reward vendors for adding features and for being first to market. These two motivations are in direct tension with the goal

* This work was funded by a Marshall Scholarship and a United Kingdom Overseas Research Student Award.

This paper has been published in the proceedings of the:
First Workshop on Quality of Protection (QoP). Milan, Italy: September 15, 2005.

of writing more secure software, which requires time consuming testing and a focus on simplicity. Nonetheless, the problems of software insecurity, viruses, and worms are frequently in the headlines; why does the potential damage to vendors' reputations not motivate them to invest in more secure software?

Vendors' lack of motivation is readily explained: the software market is a 'market for lemons' [1]. In a Nobel prize-winning work, economist George Akerlof employed the used car market as a metaphor for a market with asymmetric information [2]. In his model, buyers cannot ascertain the quality of the used cars on the market, and as a result they are unwilling to pay a premium to obtain a higher quality car. After all, why pay more for quality when you are uncertain of obtaining it? Owners of high quality cars thus become unwilling to sell them, because they cannot obtain a reasonable premium.

The software market suffers from the same asymmetry of information. Vendors may have some intuition as to the security of their products, but buyers have no reason to trust the vendors' assertions. Worse, even the vendor is unlikely to have a truly accurate picture of its software's security. As a result, buyers have no reason to pay the premium required to obtain more secure software, and vendors are disinclined to invest in securing their products.

An effective means of measuring software security could decrease the asymmetry of information and ameliorate the 'market for lemons' effect. Unfortunately, the current measures of security are a consideration of the process by which the product was made, a superficial security review of the product, or a gross consideration of its vulnerability history. In addition to being imprecise, none of these techniques are consistently reliable or particularly useful in cross-product comparison.

However, in a related domain, software engineers have invested a great deal of effort in the measurement and prediction of quality. These efforts have largely focused on three areas [3]:

1. Estimating the total number of faults in a system
2. Estimating the time-to-failure of the system
3. Quantifying the impact of design and implementation methodologies

This work has often utilized the study of faults (defects) identified during the testing and post-release lifespan of the software.

There exists a security corollary to the study of faults: the study of vulnerabilities. This work examines the feasibility of applying software *reliability* growth models to vulnerability data, which I refer to as software *security* growth modeling. Security growth modeling has the potential to provide useful predictions or metrics of security. Security growth models can produce useful and readily understandable results like the mean time to the next failure or the total estimated number of vulnerabilities in that product. These results could be used as both relative measures (with respect to competing products) and absolute measures (with respect to a desired level of assurance). However, these models are necessarily applied to noisy data and are highly dependent upon the vulnerability hunting environment. Nonetheless, security growth modeling may provide useful

quantitative insight to supplement the current approaches to assessing software security.

The next section provides an overview of reliability growth modeling and previous work. Section 3 describes the data collection challenge that is the most significant barrier to the adoption of security growth modeling. It also describes the two perspectives on vulnerability characterization that are examined here: failure and flaw. Next, in Sect. 4, I apply traditional reliability growth models to both perspectives of the same data set, and I discuss the results of this effort. In Sect. 4.3, I emphasize the need for data normalization. Section 5 highlights areas of potential future work.

2 Reliability Growth Modeling

Reliability growth models are based upon the assumption that the reliability of a program is a function of the number of faults that it contains. Such models "apply statistical techniques to the observed failures during software testing and operation to forecast the product's reliability" [4, p. 6]. As faults are identified and removed, the system will fail less frequently and hence be more reliable. These models can thus be utilized to estimate characteristics about the number of faults remaining in the system and when those faults may cause failures. They are useful for scheduling testing and for ensuring that a product meets its reliability requirements.

Unfortunately, applying reliability growth models to vulnerabilities rather than faults is impeded by a significant problem: the lack of high-quality data. The literature on reliability growth models generally assumes that they have been applied during pre-release testing and in settings where the collection of failure data was an integral part of the testing environment. Vulnerabilities are extremely unlikely to be identified as such in that stage of software development: if they are found at all, they will probably be perceived simply as faults. As a result, vulnerabilities are most often identified after the product is released—when the collection of precise data is much more difficult.

In order to be effective, reliability growth models require that the environment from which the data is obtained (usually the testing environment) must be equivalent to the environment in which the software will be utilized after deployment [5]. However, many vulnerabilities rely upon the adversary intentionally inputting abnormal data—data outside the bounds of a normal operational profile. Nonetheless, over a long period of time and the wide range of real world environments, it can be considered that the operational profile includes *all* possible input. This perspective justifies the application of these models to vulnerabilities, but it does imply that vulnerabilities may be identified more slowly than faults would be identified.

These models also require that time be normalized for testing effort. If program execution time is utilized, this assumption is readily satisfied. However, if calendar time is used then it should be normalized for the number of testers

participating, work days, holidays, *etc.* This assumption has strong implications for the usage of vulnerability data and is discussed in Sect. 4.3.

2.1 Previous Work

The ideal security metric would enable the measurement of both a product's changing security over time and its security relative to other products. Stuart Schechter noted that software producers can use a market for vulnerabilities to establish that a vulnerability in their own program is more expensive than one in a competitor's program; the vendor can thus credibly argue that its software is more secure than that of the competitor [6], [7]. I argued that a vulnerability market can be better designed as an auction; the large body of work on auction theory can then be used to optimize it [8]. Several organizations are now actively purchasing vulnerabilities, so these proposals are not unfeasible. Unfortunately, the current purchasers of vulnerabilities are not sharing pricing information, and there is no broad movement towards an open market or auction. Until such an entity or entities arise, other means of measuring software security are necessary.

Eric Rescorla has previously applied reliability growth models to post-release vulnerability data from the ICAT database [9]. In general, he found no clear trend of security growth, and he questions the social utility of publicly disclosing vulnerabilities.

However, the ICAT database is not focused on vulnerability age; as a result, it may not report all of the out-of-date versions of a program to which a vulnerability applies. This aspect of the database limits the accuracy of Rescorla's work. In previous work, I utilized a data set with full vulnerability birth and death data to challenge Rescorla's results and argue that a trend towards security growth could not yet be ruled out [10]. However, that work focused on the social utility question posed by Rescorla and the data collection process used was not well described. In particular, this process requires decisions and utilizes assumptions that have a significant bearing on the results of the analysis. This work assess two different approaches to data characterization and considers the more broad use of reliability growth models as one tool for evaluating software security.

3 Collection Technique for this Data Set

OpenBSD was selected for this study because its developers emphasize secure programming and code audit; furthermore, its entire source code and every change that has been made to it are readily accessible via internet CVS (a version control system). Version 2.2 was selected as the starting point for the data set because vulnerabilities were fixed silently in the prior two versions; this analysis relies upon the careful documentation of all vulnerabilities identified. The data set was created through the following process:

1. A list of vulnerabilities was compiled from the OpenBSD web page and the most prominent public vulnerability databases: ICAT, Bugtraq, OSVDB, and ISS X-Force.

2. The source code was examined to identify the date on which the vulnerability was repaired (the vulnerability's 'death' date).[1]
3. Prior versions of the source code were then examined until the date on which the vulnerability was introduced into the software could be identified (the vulnerability's 'birth' date).
4. Vulnerabilities were then grouped according to the version in which they were introduced. For this work, only vulnerabilities that were introduced prior to the release of version 2.2 were considered.

Although the process described above seems precise, the reality is that the data is complex and is not always readily categorizable. The most significant challenges in characterizing the vulnerabilities dealt with inclusion and uniqueness.

3.1 Inclusion

The vulnerability sources listed above included vulnerabilities that affected only specific hardware platforms or particular localizations. In the interest of universality and simplicity, vulnerabilities were included only if they were location and platform neutral (however, those specific to Intel 386 were also included, under the assumption that this platform is the most common).

In addition, the OpenBSD security page lists vulnerabilities whose inclusion stretches the definition of a vulnerability. For example, the patch description for one vulnerability listed on the OpenBSD security page is: "Improve xlock(1)'s authentication by authenticating via a pipe in an early forked process. No known vulnerability exists, this is just a precautionary patch" [11]. Although the OpenBSD security philosophy is commendable (and was the motivation for its selection as the software to model), including vulnerabilities like these has a negative impact on the models' assessment of OpenBSD's security. One way of resolving this dilemma is to include only vulnerabilities of a clearly specified and easily tested severity: *e.g.* remote root vulnerabilities. Unfortunately, assessing the risk of a potential vulnerability is enormously time consuming and risk prone. For the purposes of this analysis, no vulnerability was excluded for being unlikely or debatable. The results are thus potentially negatively biased: OpenBSD 2.2 will appear less secure than it actually is.

A similar question is posed by vulnerabilities for which the default configuration of OpenBSD is *not* vulnerable. Should those be counted? A default configuration in which most services are disabled is another commendable aspect of OpenBSD's security policy; however, in practice, many of those services will be enabled by the users. As a result, such vulnerabilities were also included in this analysis. As with the previous decision, the results are thus potentially negatively biased.

[1] If the fix was itself faulty, the date of the first effort is used rather than that of the last effort. This simplification is in accordance with most models' assumptions that flaws are fixed instantly and without introducing new flaws.

3.2 Uniqueness, or Flaw *vs.* Failure

The most difficult task was deciding upon uniqueness: whether a patch or group of patches repaired one vulnerability or multiple vulnerabilities.

OpenBSD includes some software that is maintained by third parties (*e.g.* sendmail). Those third parties often released a new version of their software that contained fixes for multiple (previously secret) security flaws. One solution is to simply count such a 'bundle' patch as repairing only one vulnerability and use the birth date of the youngest vulnerability. However, this solution will result in a positive bias and hence an inflated perception of security for the product: the models will indicate fewer vulnerabilities than actually exist and a more rapid trend towards depletion. Conversely, counting each individual security flaw in the bundle patch as a vulnerability will cause the death date of those vulnerabilities to be recorded as later than it should be: they were actually identified and repaired at some unknown date prior to the release of the bundle patch. That solution would thus bias the model away from depletion and result in an overly negative measure of security.

Similarly, individuals may find multiple related security flaws at once: either by discovering a number of security flaws of one type or by discovering a poor quality section of the code base. Often these related security flaws are remediated in the same patch; should they be considered as individual vulnerabilities or as a single, combined vulnerability?

The question of whether to consider these bundled/related security flaws as unique vulnerabilities or as a single combined vulnerability has a significant impact on the analysis. In a theoretical sense, counting them as unique is equivalent to performing the security growth modeling on *flaw* discovery data: such a data set would include dependent data points. From this perspective, each flaw is considered to be a separate vulnerability. Counting them instead as a single vulnerability is the theoretical equivalent to performing security growth modeling on *failure* data, in which every data point is independent of the others. From this perspective, a single failure initiated the discovery of multiple related security flaws. Traditionally, reliability growth models have used the times of system *failure* as their input data and require that the data points be independent.

The approach chosen has a significant impact on the analysis. In this work, the data were analyzed from both failure and flaw perspectives. Table 1 shows the differing vulnerability counts when each approach is used. The first row shows the number of vulnerabilities discovered per year when related and bundled vulnerabilities are grouped (the failure/independent perspective). The second row shows the number of vulnerabilities discovered per year when vulnerabilities are considered individually (the flaw/dependent perspective). For the flaw perspective, each of those unique–but possibly dependent–vulnerabilities was used as a data point, thus increasing the total number of data points considered. Note that the data for 2002 covers only the first five months of the year.

Table 1. Vulnerabilities identified in OpenBSD 2.2 from 1998-01 – 2002-05†

Perspective	1998	1999	2000	2001	*2002*‡	Total
Treated as failures (only independent data points)	19	17	17	13	*2*	68
Treated as flaws (dependent data points included)	24	18	22	13	*2*	79

†No vulnerabilities were found in December 1997, the first month that version 2.2 was available.
‡The first five months of 2002.

4 Results

4.1 Rate of Vulnerability Detection

I analyzed both the failure- and flaw-perspective data sets with seven time-between-faults reliability growth models.[2]

The data indicate the number of days that elapsed between the identification of faults. The mean, median, and standard deviation for the failure-perspective data are: 23.7, 13.5, and 28.0. For the flaw-perspective data: 19.8, 7.0, and 26.54. For both data sets, the minimum was 0 and the maximum was 126.

Three models were applied successfully to the failure-perspective data set; these three models had acceptable bias, noise, trend, and goodness-of-fit results. Table 2 shows the pertinent applicability results. For each quantitative result, that model's ranking with respect to the other two models is shown in parentheses. The first row shows bias, as determined by a μ-plot; this measure assesses the absolute predictive accuracy of the models. The noise and trend results in the second and third rows are useful primarily to ensure that the predictive accuracy indicated by the μ-plot results was not due to opposing trends of inaccuracy canceling each other out on the average. The prequential likelihood values of the three models, shown in row four, are used to assess the relative accuracy of the models with respect to each other. Overall, Musa's Logarithmic model was the most accurate and was ranked first (1).[3]

Table 2. Applicability results for models applied to the failure-perspective data

	Successful Models		
Statistic	Musa's Logarithmic	Geometric	Littlewood/Verrall (L)
Bias (μ-plot)	0.12 (1)	0.13 (2)	0.18 (3)
Noise	0.31 (1)	2.39 (2)	2.44 (3)
Trend (y-plot)	0.20 (3)	0.18 (2)	0.14 (1)
Prequential Likelihood	148.35 (1)	150.23 (2)	150.50 (3)
Overall Rank	(1)	(2)	(3)

[2] The SMERFS[3] reliability growth modeling tool was used to assess the models [12].
[3] For a more detailed explanation of the acceptability tests, see [13]

None of the seven models were successfully applied to the flaw-perspective data. Each model applied to this data set failed one of four tests: bias, trend, noise, or goodness-of-fit. This failure is not surprising: reliability growth models require that their data points be independent: the flaw-perspective data included vulnerabilities whose discovery was clearly dependent upon the recent discovery of a similar vulnerability. As a result, it seems likely that failure-perspective analysis is a superior method of considering vulnerabilities; it has a sound theoretical basis and the attempt to model the quantitative data was much more successful with this approach. However, data filtering, regardless of the theoretical justification, is always suspect. I am gathering three more years of data in order to verify the accuracy of these results.

Table 3 displays the various estimates produced by the models successfully applied to the failure-perspective data. The intensity is the expected number of vulnerabilities per day. Rows one and two display the intensity at the first and last day of the analysis. The purification level, shown in row three, is a normalized estimate of how vulnerability-free the program is at the end of the period covered by the data set. A purification level of one would indicate a program entirely free of vulnerabilities. The purification level formula used here is undefined for infinite-failure models like Littlewood/Verrall Linear; however, alternative formulations of purification level can be used for these models [14]. The fourth row displays the current Mean Time To Failure (MTTF), the expected number of days before the next vulnerability is identified.

Table 3. Estimates made by the successful models using the failure-perspective data set

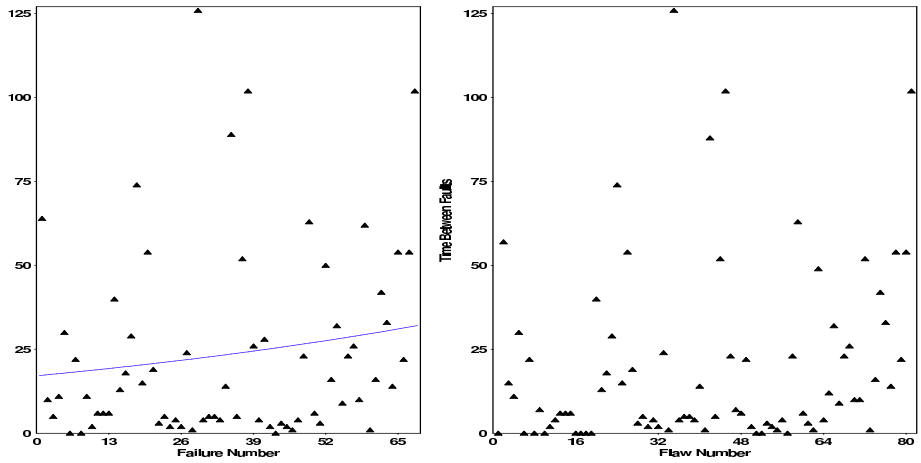
	Successful Models		
Statistic	Musa's Logarithmic	Geometric	Littlewood/Verrall (L)
Initial Intensity	0.059	0.062	0.066
Current Intensity	0.031	0.030	0.030
Purification Level	0.584	0.505	N/A
Current MTTF	42.5	33.1	33.8

Figure 1 shows both the failure-perspective data set (left) and the flaw-perspective data set (right). For the former, the successfully fitted Musa's Logarithmic model is shown superimposed over the data set; this model was ranked as most accurate of the three successful models. For the latter data set, no models were successfully applied, so the data points alone are displayed.

4.2 Trend Analysis

Reliability growth models assume an eventual trend in which the rate of vulnerability detection decreases over time. One way to test for such trends is through the use of a Laplace test [5]. The calculated Laplace Factors for each data set

Fig. 1. Time-between-fault data sets

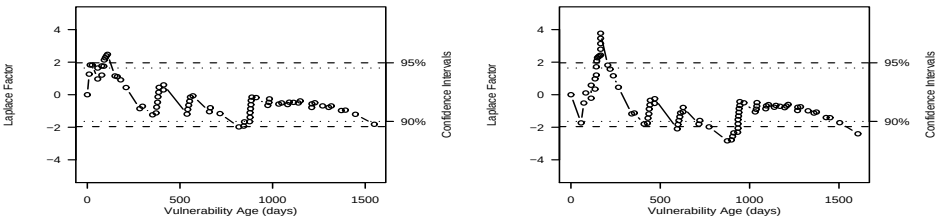


(a) Failure-perspective data with fitted Musa's Logarithmic model

(b) Flaw-perspective data

are shown in Fig. 2. Values below 0 indicate a trend towards decreasing rate of vulnerability detection. However, only values below -1.96 indicate that trend within a 95% confidence level for a two-tailed test (-1.64 for a 90% confidence level). The null hypothesis, that the data exhibits no trend, cannot be rejected for the failure-perspective data set. The flaw-perspective data set shows a more clear trend towards decreasing rate of vulnerability detection. Again, however, the null hypothesis of no trend cannot be ruled out for large periods of time. Both data sets have an initial period in which the rate of vulnerability detection increased. This initial increase is likely caused by the sudden increase in users and environments of use after the software was released; it suggests that an S-shaped reliability growth model may be most appropriate. However, none of the models with acceptable predictive accuracy were of this category.

Fig. 2. Trend Analysis



(a) Failure-perspective data

(b) Flaw-perspective data

4.3 Data Normalization

The results indicate that a decreasing rate of vulnerability detection cannot simply be assumed for this data set. Why would the rate of vulnerability detection have increased or stayed constant? One possible answer is that the effort invested in vulnerability discovery during the time period covered in this study increased: more individuals searched for vulnerabilities or those who searched grew more capable of finding vulnerabilities.

As discussed in Sect. 2, one of the underlying assumptions of all reliability growth models is that the data is normalized for effort. The data for the time necessary to find a vulnerability should ideally be the execution time; if such data is not available, the time should be the skill-equivalent person hours. Unfortunately, the data available on vulnerabilities does not include the number of individuals examining that software, much less their relative skill.

This data set thus cannot provide an accurate characterization of the 'true' security of the product (*i.e.* the number of unknown vulnerabilities in the product). The time period from which data was collected, 1997-12-01 to 2002-05-31, witnessed an explosion of interest in computer security and the identification of vulnerabilities. It thus seems likely that many more individuals were searching for vulnerabilities in 2002 than in 1997, but the data used here does not take this change into account. As a result, the the trend analysis and the estimate of the total number of vulnerabilities discussed below may not be an accurate characterization of the underlying product: they are probably conservative and thus characterizes the product as less secure than is actually the case.

The three reliability growth models in Sect. 4.1 have demonstrated acceptable predictive accuracy, but the analysis in Sect. 4.2 cannot rule out the possibility of no significant change in the rate of vulnerability detection. The successfully applied reliability growth models may be accurately characterizing the decreasing rate that appeared in the trend analysis towards the end of the study. Although this data set lacks the normalization discussed above, reliability growth models can still provide insight into the changing rate of vulnerability detection over time. At the very least, these models can describe that rate given the current vulnerability hunting environment. However, the discrepancies between the reliability growth results and the trend analysis indicate that more data is needed before a confident assessment of the system can be made.

5 Future Work

This work highlights five interesting areas for further research: normalize the data for effort, examine the return on security investment, utilize more sophisticated modeling techniques, and combine vulnerability analysis with traditional 'software metrics.'

As discussed in Sect. 4.3, the data set used is not normalized for effort: the skill of and number of individuals searching for vulnerabilities. Unfortunately, OpenBSD does not release usage figures; because it is often used as a server

operating system, other available sources of usage data are also inadequate (*e.g.* the proportion of web browsing done from OpenBSD). Moreover, the number of users of a product is not necessarily a useful correlate to the number of individuals searching for vulnerabilities in the product. One area of future work is to find a proxy for effort, at least with respect to the number of individuals searching for vulnerabilities. One possible proxy is the relative numbers of individuals posting to 'full-disclosure' security lists like Bugtraq and Full Disclosure. Although finding an exact measure of effort would be prohibitively difficult, a relative measure would still be useful: *e.g.* there were twice as many individuals searching for vulnerabilities in 2000 as there were in 1998.

Another direction for this research is to examine the return on investment for secure coding practices. Do models fitted to Microsoft's post-2002 secure coding initiative indicate that it is producing results?

An additional path forward is to employ more sophisticated techniques for modeling security growth. The reliability growth literature is rich with means of improving models' accuracy. Finally, vulnerability analysis can be combined with traditional 'software metrics:' metrics that attempt to measure the size, complexity, *etc.* of a program. This line of research might lead to other fruitful measurements or predictors of security.

6 Conclusion

Software engineering provides useful tools for the analysis and potential measurement of software vulnerabilities. In this work, 54 months of vulnerability data were gathered for the OpenBSD operating system. The source code was examined to ascertain the exact dates when the vulnerability was first added to the code and when it was repaired.

The data collection process was complex, and I struggled to find rules for data characterization that covered all possible situations. For OpenBSD, collection difficulties centered around inclusion (when is a defect considered a vulnerability) and uniqueness (when do a number of defects qualify as one vulnerability and when do they qualify as multiple vulnerabilities). Two different characterization criterion were analyzed here: as expected, the reliability growth modeling was only successful when considering the data set that excluded dependent data points.

Three of the seven reliability growth models tested were found to have acceptable one-step-ahead predictive accuracy for the set of independent data points. Musa's Logarithmic model was the model ascertained to be most accurate of those three; it estimated that the mean time to failure at the end of the study is 42.5 days and 58.4% of the estimated total vulnerabilities in the product have been identified. Together, these estimates could serve as both a useful relative and absolute measure of the security of the product.

However, a trend analysis cannot rule out the possibility that vulnerabilities are being detected at a constant overall rate, which casts doubt on the results produced by the reliability growth models. If these results are a more accurate

reflection of the system, the lack of a decreasing vulnerability detection rate may be due to an increase in the amount of effort invested in finding vulnerabilities during the course of the study. Reliability growth models and trend analysis are designed for data in which the amount of effort invested in finding vulnerabilities is constant. Unfortunately, no information is available on the growth in the number of individuals searching for vulnerabilities and the effort they invested; as a result, the data set cannot be normalized to take this information into account.

The results of this analysis are thus inconclusive. More data is needed before a definitive assessment can be made of the rate of vulnerability detection in OpenBSD. This problem highlights the main challenge in using software engineering tools to analyze vulnerabilities: the significant effort required in order to collect accurate data and the lack of availability of important information.

Despite these difficulties, this analysis has shown that software engineering tools can provide useful insight into software vulnerabilities. Security growth modeling, the application of reliability growth models to vulnerabilities, can build upon a long tradition of software engineering work, adapting that work as appropriate. If the technique increases in popularity, data collection could be readily incorporated into the vulnerability remediation process. Better data collection would, in turn, result in more accurate and more useful models.

References

1. Anderson, R.: Why information security is hard - an economic perspective. In: 17th Annual Computer Security Applications Conference. (2001) New Orleans, LA, USA.
2. Akerlof, G.A.: The market for 'lemons': Quality uncertainty and the market mechanism. The Quarterly Journal of Economics **84** (1970) 488–500
3. Fenton, N.E., Neil, M.: A critique of software defect prediction models. IEEE Transactions on Software Engineering **25** (1999) 675–689
4. AIAA/ANSI: Recommended Practice: Software Reliability. ANSI (1993) R-013-1992.
5. Lyu, M.R., ed.: Handbook of Software Reliability Engineering. McGraw-Hill (1996)
6. Schechter, S.: Quantitatively differentiating system security. In: Workshop on Economics and Information Security. (2002) Berkeley, CA, USA.
7. Schechter, S.: How to buy better testing: Using competition to get the most security and robustness for your dollar. In: Infrastructure Security Conference. (2002) Bristol, UK.
8. Ozment, A.: Bug auctions: Vulnerability markets reconsidered. In: Workshop on Economics and Information Security. (2004) Minneapolis, MN, USA.
9. Rescorla, E.: Is finding security holes a good idea? In: Workshop on Economics and Information Security. (2004) Minneapolis, Minnesota.
10. Ozment, A.: The likelihood of vulnerability rediscovery and the social utility of vulnerability hunting. In: Workshop on Economics and Information Security. (2005) Cambridge, MA, USA.
11. OpenBSD: OpenBSD 2.8 errata, 014: Security fix (2000) `http://www.openbsd.org/errata28.html#xlock`.

12. Stoneburner, W.: SMERFS (Statistical Modeling and Estimation of Reliability Functions for Systems) (2003) `http://www.slingcode.com/smerfs/`.
13. Abdel-Ghaly, A.A., Chan, P.Y., Littlewood, B.: Evaluation of competing software reliability predictions. IEEE Transactions on Software Engineering **12** (1986) 950–967
14. Tian, J.: Integrating time domain and input domain analyses of software reliability using tree-based models. IEEE Transactions on Software Engineering **21** (1995) 945–958