Towards a Bayesian Approach in Modeling the Disclosure of Unique Security Faults in Open Source Projects*

Prasanth Anbalagan[1] Mladen Vouk[2]
Department of Computer Science , North Carolina State University, Raleigh, NC 27695, USA
[1]panbala@ncsu.edu [2]vouk@ncsu.edu

Abstract

Software security has both an objective and a subjective component. A lot of the information available about that today is focused on security vulnerabilities and their disclosure. It is less frequent that security breaches and failures rates are reported, even in open source projects. Disclosure of security problems can take several forms. A disclosure can be accompanied by a release of the fix for the problem, or not. The latter category can be further divided into "voluntary" and "involuntary" security issues. In widely used software there is also considerable variability in the operational profile under which the software is used. This profile is further modified by attacks on the software that may be triggered by security disclosures. Therefore a comprehensive model of software security qualities of a product needs to incorporate both objective measures, such as security problem disclosure, repair and, failure rates, as well as less objective metrics such as implied variability in the operational profile, influence of attacks, and subjective impressions of exposure and severity of the problems, etc. We show how a classical Bayesian model can be adapted for use in the security context. The model is discussed and assessed using data from three open source software project releases. Our results show that the model is suitable for use with a certain subset of disclosed security faults, but that additional work will be needed to identify appropriate shape and scaling functions that would accurately reflect end-user perceptions associated with security problems.

1 Introduction

Traditionally software reliability models observe software failures in the context of operational use of a system. Preferred time in that context is software execution time or inservice time (e.g., [16], [8]) rather than calendar time. In contrast, security is often viewed from the perspective of security problems (or faults) rather than failures, and from the perspective of the attackers. Security researchers often focus on vulnerabilities (i.e., faults) independently of the operational profile of the software that harbors them [13, 1, 9, 21]. This is in many ways understandable. Affected parties maybe reluctant to report security breaches (failures), so often disclosure of vulnerabilities is the only way to gain an insight into the security profile of a product. We believe that operational use of a software product needs to be taken into account explicitly when analyzing its security profile.

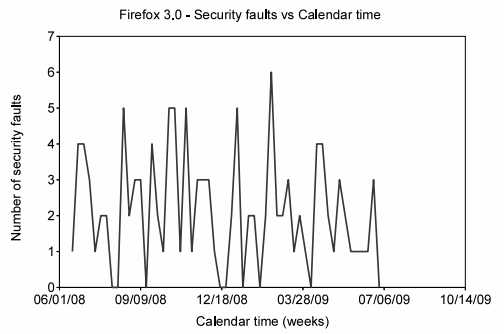

Figure 1: Security faults vs Calendar time

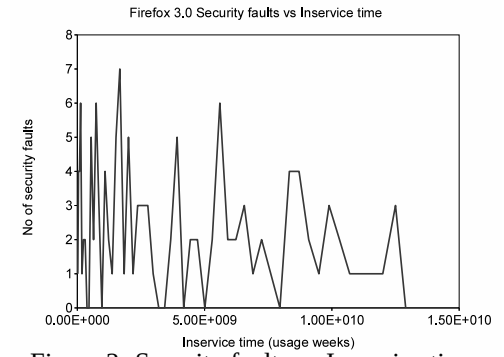

Figure 2: Security faults vs Inservice time

For example, Figures 1 and 2 show the number of secu-

*This work is supported in part by the U.S. Army Research Office (ARO) under grant W911NF-08-1-0105 managed by NCSU Secure Open Systems Initiative (SOSI).

rity faults disclosed for Firefox 3.0 in a calendar week and the number disclosed during an inservice week. It is interesting to observe that that the number of security faults disclosed per unit time remains approximately constant when viewed from the calendar perspective. It appears to decrease when viewed from the perspective of inservice time. The latter view may better reflect the security quality of the software as a whole, since it focuses on the disclosure of security faults along with the growth in the number of users. In real life operational profile of a software product can include not only legitimate use of that software, but also attacks by hackers. The latter may intensify (and change the operational profile) after a vulnerability is disclosed. When a disclosure is accompanied by a fix or a work-around, end-user perceptions of the level of security of the product may improve, i.e., security reliability of the product may appear to increase. When disclosures are not directly accompanied by instant repair of the problem, end-user perceptions may be less favorable despite the fact that in actuality the probability that the product will fail due to a security breach may not have changed in a particular environment. Therefore we distinguish two groups of security disclosure

1. those that are accompanied by fixes, distribution of weekly fixes by a vendor, or those where the end-user, due to the nature of the vulnerability, can implement an instant fix, problem avoidance or work-around solution.

2. those that require an external fix, and have the fix follow a measurable time (μ) later, or possibly never.

In [4], we observed that one facet of how a vulnerability is exploited is whether a user is 1) needs to be deceived into interacting with (helping) the attack mechanism to effect the exploit, or 2) the vulnerability is not dependent on interactions with the user. We called the former "voluntary" exploit problems or faults [4]. We call the latter "involuntary" exploit faults. Security faults may also be disclosed as a result of non-security failures. For example, consider the Firefox vulnerabilities under MFSA 2009-071. Developers observed crashes in the Firefox browser engine during the normal usage of the application. Some of these crashes were traced back to memory corruptions which one could potentially exploit. Since inputs from the normal operational use of the system is a component in uncovering security faults, ignoring operational use in security analyses may not be appropriate. In this work we examine disclosure of security vulnerabilities in open source software from several perspectives, including a subjective one. We do that using data for two releases of Firefox and one release of SeaMonkey.

The rest of the paper is organized as follows. In Section 2 we discuss the subjective impacts of the security disclosures, and adapt a classical software reliability engineering Bayesian model to this context. In section 3 we assess the model using experimental information. Section 4 discusses related work and Section 5 concludes the paper.

2 Modeling Disclosure related Beliefs

2.1 Sources of uncertainty

Software tends to come with faults. As it ages some of those faults are detected and removed, but other faults could be introduces during that process as well. We consider security vulnerabilities as a subset of the general pool of faults. The difference, however, is that security faults imply malicious intent and exploitation that may deliberately amplify a relatively innocuous deficiency in software which, on its own, may not cause a software failure. If we accept that a certain fraction of the operational profile of network-based software are attacks or attempts to exploit its vulnerabilities, then a software product (running on many installations and with many users) will experience disclosure of its vulnerabilities in a relatively random fashion (e.g., [6]). Of course, attacks may intensify after a problem disclosure, and that may change both the actual probability that a machine will suffer a security failure and the perceived probability. On the other hand, in the case of closed source projects, disclosures of security problems may occur at regular intervals - often after fixes for the problem are available (e.g., [15]).

While software may originally come with faults, once they are disclosed perceptions of the user can change. How that perception, belief in the security quality of the software, changes may depend on several factors. First of all, disclosure of unique faults does not really say anything about the failures (or exploits) due to those vulnerabilities, nor how often those occur. Therefore, just disclosing a problem without some additional action may only worsen the perception an end-user may have of that software. If disclosure is accompanied by a fix, then in many situations perceptions of the users will become more positive (i.e., software security reliability perception may grow).

There is also the issue of trust. Open source projects, particularly when the community is large, come with a certain belief that disclosures are made in earnest and as soon as issues are discovered. If, in addition a disclosed vulnerability is of the voluntary class, or somehow else can be mitigated (e.g., blocking of certain ports to external access), or an actual fix is available, then this is equivalent to failure reporting with instant fault repair in traditional software reliability engineering. In general the trust, or belief, in the software from the end-user security perspective can increase, remain the same, or decrease with every vulnerability disclosure.

One can discuss the probability from a frequentist or from a Bayesian perspectives [10, 12]. For example, given some assumptions, the rate of disclosing a unique security

fault in a time period can be estimated as the ratio between the number of disclosures and the time over which those disclosures have been made This can then be translated into probability that a disclosure (or a security failure) will be observed over a certain time period in the future, etc.

However, typically software projects of interest have a large number of users with perhaps widely varying operational profiles. In that case it may not be justifiable to just use frequencies and point estimates to assess system behaviors, especially if the nature of the faults and failures is such that there is a considerable variability in the interpretation of their impact, or even whether an impact can occur or not. Security failures and faults fall into that domain. We believe that in this case analysis of the disclosures of the problems, through conditional probability distributions, i.e., a Bayesian view [11]., may provide a better envelope for estimating their possible impact. Bayesian models have been used with success in the past to describe and predict behavior of general class of software failures ([12, 11, 10]), but their application to software security has been very limited.

One thing to remember is that while security faults are in many ways similar to non-security faults (i.e., they are due to an omission in the specifications, or are a physical defect in the code, a user error, etc.) they also exhibit many behavioral differences. For example, they may not manifest in physical failure of the system but only in more ephemeral (but still unintended) behaviors such as system slow down, unauthorized access to the system and its content, they may not manifest unless there is a correlated series of inputs, including perhaps end-user cooperation (voluntary failures), etc.. Thus focusing on disclosures may be more telling than focusing on the actual security breaches (or actual security failure intensity) so long as one remembers that not every disclosed (potential) vulnerability is actually exploitable. Furthermore, security failures often require operational conditions and inputs which are considerably different from the operational profile envisioned for the software under considerations. This, and the unknown factor of attacker behaviors (timing, interest, location, etc.) can drastically alter operational profile of a system or a group of systems.

2.2 Data

In this paper, we used data for Firefox 3.0, Firefox 3.5 and SeaMonkey 1.1. The data, including information on the number of downloads, number of security faults, report time, and repair time, were collected using [3] from Mozilla[1] pubic repositories. We considered only security faults disclosed after the release of each product. We assumed that the number of downloads is proportional to usage of the product in the field. This allowed us to estimate an upper bound on the inservice time as the product of the number of systems using the product and the calendar time since product release.

[1] http://www.mozilla.org/security/known-vulnerabilities/

2.3 Model

We use a variant of Littlewoods approach to Bayesian modeling [12, 11, 10], except we apply the reasoning to analysis of software security vulnerability disclosures.

A obvious source of uncertainty is random disclosure of unique security faults during the operational use of the software. Typically such random events are assumed to follow a Poisson process [16]. But security researchers [22, 17] have questioned the validity of making such assumptions. This has prompted us to check on this assumption.

To test for a Poisson process, it is necessary to establish whether inter arrival times of unique security fault disclosures follow an exponential distribution, and whether the number of unique security faults disclosed in a fixed time periods follow a Poisson distribution [20].

First we tested for the exponential distribution of inter arrival times using the chi-square (χ^2) goodness of fit test. The chi-square test is used to test if the given data follows a specified distribution. The null hypotheses is stated as "H0: The inter-arrival times of unique security faults follow an exponential distribution". The chi-square test is applied to data grouped in to bins. Since there is no optimal size for the bin [23], we selected five random bin sizes to test for similarity among the results. The bin sizes are time intervals of (4,000,000,000), (6,000,000,000), (8,000,000,000), (10,000,000,000), and (12,000,000,000) inservice minutes. We used Firefox 3.0 data to test if the inter arrival times measured in execution times followed an exponential distribution. Table 2 shows the results of the test.

To illustrate, consider bin size of 6,000,000,000. Using 0.01 as the level of significance, the critical value of χ^2 with 3 degrees of freedom was found as 11.35. The test statistic for this bin size was calculated as 3.22. Since χ^2=3.22<11.35, we do not reject the null hypotheses, i.e., there is not enough evidence to conclude that the inter-arrival times of unique security faults do not follow an exponential distribution. Similarly, we used the chi-square (χ^2) goodness of fit test to verify if the number of unique disclosures in fixed time periods follow a Poisson distribution. The null hypotheses is stated as "H0: The number of unique security faults disclosed per time period follows a Poisson distribution". We selected five random bin sizes to test for similarity among the results. The bin sizes are time intervals of (4,800,000,000), (4,500,000,000), (4,200,000,000), (2,400,000,000), and (1,500,000,000) execution minutes. Table 3 shows the results of the test.

Consider the bin size of 4,500,000,000 inservice minutes. Using 0.01 as the level of significance, the critical

Table 1: Chi-square (χ^2) test for Exponential Distribution

Level of Significance(α) = 0.01					
No	Bins	Df	Critical Value	Test Statistic	Result
1	4,000,000,000	5	15.08	8.81	Fail to reject
2	6,000,000,000	3	11.35	3.22	Fail to reject
3	8,000,000,000	2	9.21	2.71	Fail to reject
4	10,000,000,000	2	9.21	5.39	Fail to reject
5	12,000,000,000	1	6.64	1.43	Fail to reject

value of χ^2 with 4 degrees of freedom was found as 13.28. The test statistic for this bin size was calculated as 6.89. Since χ^2=6.89<13.28, we do not reject the null hypotheses, i.e., there is not enough evidence to conclude that the number of unique security faults disclosed per time period does not follow a poisson distribution. Similar results were obtained for Firefox 3.5 and SeaMonkey 1.1 releases.

Similar to Littlewood's model [12], we now focus on the time-to-unique-security-fault disclosure distribution, and time-to-repair of such faults. Hence the data for this model is a sequence of execution time intervals, i.e., the time interval between successive disclosures of unique security faults measured in inservice times, and time to repair those faults. Since we have established that the inter arrival time of unique security fault reports is exponentially distributed, then assuming we know the disclosure rate of unique security faults, its probability distribution function is given by

$$pdf(t_i|\lambda_i) = \lambda_i e^{-\lambda_i t_i} \quad (1)$$

where t_i is the execution time between the disclosure of unique security faults and λ_i is the disclosure rate of unique security faults at disclosure event i.

Unfortunately, disclosure rate is not a known quantity to start with. Furthermore there is a considerable amount of uncertainty attached to this value due, among other things, to possible attacker induced changes in the operational profile, and other perhaps perception-based decisions. Hence we treat the disclosure rate as a random variable.

Figure 3 shows a very simplified disclosure model adapted from the more comprehensive model discussed in [4]. λ denotes problem disclosure rate (e.g., problems per unit time), and μ problem repair rate (problems repaired per unit time. Software is in the Good/Hidden state during normal operation (hidden refers to latent or hidden faults that have not yet been disclosed). When a security fault is disclosed it transitions into Disclosed state and stays there until the fault is repaired (fixed), after which it transitions back to the Good/Hidden state.

Faults that are disclosed but are not repaired (μ = 0, or $\mu >> \lambda$) are likely to increase concerns about the system. A relatively constant fault discovery rate (λ) over longer periods of time may do the same since it may imply either a software that had new faults injected, or a software that has a very large pool of problems, or perhaps a software where some of the faults are kept secret. Usually in open source projects security faults are disclosed immediately upon discovery and then they are fixed. If repair time is short (e.g., $\mu > \lambda$) faults will not accumulate, and that is good. Some disclosures, such as those in the voluntary category allow the end-user to mitigate their possible impact immediately making μ effectively infinite or at least much larger than λ. But security faults may also be kept secret upon discovery, then fixed and disclosed along with the fix. In that case public μ is technically infinite or at least much larger than λ. This is typically the case in closed source projects. This may increase the confidence of a user if one believes that the fix is a good one. On the other hand, there real value of μ usually is not know in those cases, and so it is not clear how long end-users have been exposed to the danger. If at the same time λ has been relatively constant over extended time periods the confidence may decrease because one may become apprehensive about what other security problems may have been concealed from a user and currently leave the user exposed to attacks.

Open source proponents favor disclosure of security faults since they believe in transparency, and that public disclosures allow issues to be addressed quickly, allow users to take precautions against any exploits [4], and in general guard against security through obscurity approach. If the rate of public disclosures of security faults is a decreasing function of time, one also needs to make certain that the associated repair time is sufficiently short that the actual pool of security faults in a software is a decreasing function of time.

Lets consider the case where on the average $\lambda_i < \lambda_{i-1}$ for event i. We would also need to insure that the average repair rate is large enough that there is no accumulation of faults. If there is uncertainty about the values of λ, we at least would like to ensure that it is probable that the public disclosure of a security faults, and its repair action rate, increases the confidence of a users.

Let us, for simplicity, focus on a sub-set of security faults those that fall into the voluntary class, and those that come with fixes. Then in both cases, from the model perspective

Table 2: Chi-square (χ^2) test for Poisson Distribution

Level of Significance(α) = 0.01					
No	Bins	Df	Critical Value	Test Statistic	Result
1	1,500,000,000	3	11.35	7.06	Fail to reject
2	2,400,000,000	3	11.35	6.54	Fail to reject
3	4,200,000,000	4	13.28	2.69	Fail to reject
4	4,500,000,000	4	13.28	6.89	Fail to reject
5	4,800,000,000	4	13.28	4.38	Fail to reject

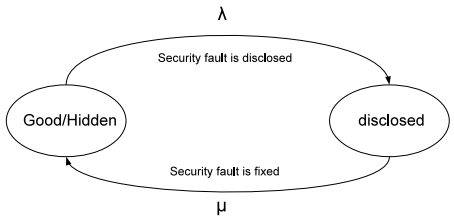

Figure 3: Disclosure Model

μ=∞, or at least $\mu >> \lambda$. If the mean-time-to-repair is zero or a user is not certain in the fix that has been provided, then the governing uncertainty is that in λ. If a user is certain that the fault has indeed been fixed, then for λ, this can be stated as

$$P(\lambda_i < l) \geq P(\lambda_{i-1} < l), for\ all\ l, i \tag{2}$$

Let the distribution function of λ_i be denoted by G(l,i,α), where α is a parameter or vector of parameters. Then,

$$G(l, i-1, \alpha) \leq G(l, i, \alpha) \tag{3}$$

This requires us to choose a parametric family that satisfies the ordering of the distribution functions. Similar to Littlewood et al. [14, 12], we use Gamma distribution to represent the disclosure rate. Note that the scaling function (β) in the Gamma distribution need to be a function appropriate to satisfy the above ordering of the distribution functions. For example, it could be monotonically increasing to satisfy the increasing order of the distribution functions. Since we deal with voluntary security faults, we use a constant beta function. This can be reasoned as follows. Assume the developer does not fix the fault immediately but starts working on the fix at his own perusal. While the fault is still being fixed the user has a workaround to avoid any exploit. One may argue that the fault is not yet fixed and may decrease the confidence of the user. But from the perspective of a user whose belief in the open source community is sufficiently greater to overcome any pessimism at the disclosure, the confidence increases considering that the user has a workaround to avoid exploit and that the developer will eventually fix the fault. This is also evident from our data since all faults were eventually fixed.

We make the following assumptions

1. The operational use of the software and the intervention of attackers together form the operational environment.

2. The disclosure of unique security faults is random and follows a Poisson process.

3. The disclosure rate of unique security faults at any time is a random variable and is assumed to follow a Gamma distribution.

4. The security faults are fixed immediately upon disclosure and no new security faults are introduced by the repair action.

In the following, for simplicity, neither λ, nor the Gamma shape (α) or scaling (β) parameters are shown as functions of time and event counts. This may not actually reflect the reality. Security faults often are not fixed immediately. Therefore we would need to account for the actual repair time in the fault correction process. Furthermore, operational profile may change and λ, α and β and their distributions are very likely functions of time, and certainly of perceptions. We plan to address this limitation at a later date.

Consider that we observe the software when a total of τ execution minutes have passed and n unique security faults have been disclosed. We are interested in the time to next disclosure of a unique security fault. The conditional distribution is given by,

$$pdf(\lambda |\ n\ security\ faults\ disclosed\ in\ (0, \tau))$$

$$= \frac{(Pr(n\ faults\ disclosed\ in\ (0,\tau))|\lambda).pdf(\lambda)}{\int (Pr(n\ faults\ disclosed\ in\ (0,\tau))|\lambda).pdf(\lambda).d\lambda}$$

$$= \frac{\frac{\lambda\tau}{n!}.e^{-\lambda\tau}.pdf(\lambda)}{\int \frac{\lambda\tau}{n!}.e^{-\lambda\tau}.pdf(\lambda).d\lambda} \tag{4}$$

Since we have assumed that the disclosure rate follows a Gamma distribution, we have

$$pdf(\lambda) = \frac{\beta^{\alpha}.\lambda^{\alpha-1}.e^{-\beta\lambda}}{\Gamma(\alpha)} \quad (5)$$

Using (3) in (2) we get,

$$pdf(\lambda|\ n\ security\ faults\ disclosed\ in\ (0,\tau))$$

$$= \frac{\frac{\lambda\tau}{n!}.e^{-\lambda\tau}.\frac{\beta^{\alpha}.\lambda^{\alpha-1}.e^{-\beta\lambda}}{\Gamma(\alpha)}}{\int \frac{\lambda\tau}{n!}.e^{-\lambda\tau}.\frac{\beta^{\alpha}.\lambda^{\alpha-1}.e^{-\beta\lambda}}{\Gamma(\alpha)}.d\lambda}$$

$$= \frac{(\beta+\tau)^{(\alpha+n)}.\lambda^{\alpha+n-1}.e^{-\lambda(\beta+\tau)}}{\Gamma(\alpha+n)} \quad (6)$$

which is gamma distributed with parameters ($\alpha+n$,$\beta+\tau$). We had already mentioned that,

$$pdf(t|\Lambda=\lambda) = \lambda e^{-\lambda.t} \quad (7)$$

From (5), the unconditional time-to-next-security-fault disclosure distribution is,

$$pdf(t) = \int_0^{\infty} pdf(t|\Lambda=\lambda).pdf(\lambda|\ n\ security\ faults\ disclosed\ in\ (0,\tau)).d\lambda$$

$$= \frac{(\beta+\tau)^{(\alpha+n)}.(\alpha+n)}{(\beta+\tau+t)^{(\alpha+n+1)}} \quad (8)$$

which is a Pareto distribution. Next, we are interested in computing the security fault disclosure rate. The failure rate of a program is computed from the reliability of the program, i.e.,

$$\begin{aligned} R(t) &= P(T>t) \\ &= 1-P(T<=t) \\ &= 1-cdf(t) \end{aligned} \quad (9)$$

$$\lambda(t) = \frac{-R'(t)}{R(t)} \quad (10)$$

We use a similar approach in computing the security fault disclosure rate. In our case, we adapt the reliability of the program to represent the probability that there is no disclosure of unique security faults up to time t. Note that t is measured from the τ point on, so these equations actually reflect piecewise nature of the problem. Although, this may not be the actual reliability of a program, we use it to compute the disclosure rate of unique security faults. Since the time to next disclosure follows a Pareto distribution, we get,

$$R(t) = \frac{(\beta+\tau)^{(\alpha+n)}}{(\beta+\tau+t)^{(\alpha+n)}} \quad (11)$$

$$\lambda(t) = \frac{\alpha+n}{(\beta+\tau+t)} \quad (12)$$

Similar to Littlewood's approach [14, 12], we have obtained the unconditional unique security fault disclosure rate for an open source software for which a total of "n" unique security faults have been disclosed in a total execution time of τ.

3 Evaluation

We use security data from Firefox releases 3.0 and 3.5, and from SeaMonkey release 1.1 to discuss the Bayesian model in the context of experimental information.

We continue to use inservice time as the primary exposure metric. Figures 4, 6, and 8 show the number of security faults reported for Firefox releases 3.0, 3.5 and SeaMonkey release 1.1 respectively. The horizontal axis shows the inservice time in inservice weeks - an estimate of the number of operational systems multiplied by the number of weeks in operation. For each week we have taken the data on how many software downloads were recorded. We made the assumption that all downloaded systems were installed and activated, and then used that information and the information from previous weeks to compute an upper bound on the cumulative number of systems in service that week. The vertical axis shows the number of security faults disclosed in each week. From the figures, we observe that the number of security faults reported over the inservice time either decreases(Figure 6), or is approximately constant (Figures 4 and 8). Figures 5, 7 and 9 show the fault disclosure rates. Figure 16 shows an example of disclosure and repair rate on a calendar perspective.

Table 3 summarizes some of the more interesting statistics about the three releases. These statistics are over the whole observed period noted in the table. The total number of security problems is relatively small. This is particularly notable since there were millions of downloads of these systems during the observed period. We see that for most part repair processes are keeping up with problem reporting (e.g., Figure 16), but that in some cases issues took a very long time to fix (from the data on average disclosures and repair and their standard deviation). On the average, one is exposed to an unresolved problem for 3 to 8 weeks. This can be a long time if the system has a high probability of being attacked. The good news is that about 30% of the observed problems were of the "voluntary" type and

Table 3: Security Faults Statistics

Project	Firefox 3.0	Firefox 3.5	SeaMonkey 1.1
Number of users at the end of the period	450,459,683	311,992,340	2,309,549
Period of interest (weeks)	54	50	50
Number of security faults reported	125	57	69
Average number of disclosures per calendar week	2.27	1.12	1.23
Standard deviation	1.83	1.27	1.32
Average number of fixes per calendar week	2.16	1.37	2.75
Standard deviation	9.73	3.59	11.59
Average repair time (weeks)	7.98	2.79	3.91
Standard deviation	10.26	2.71	4.94
Maximum repair time (weeks)	55.86	10.06	24.19
Minimum repair time (weeks)	0.01	0.04	0.01

pro-active mitigation of the issue by the end-user is thus feasible.

So how does this get captured by a model. The current Bayesian model assumes instantaneous fault repair. In about 30% of the cases (voluntary problems) that is feasible [4], but that is not the case in general. However, problem repair rates are at least keeping up with issues, although with some delay. Figures 10, 11 and 12 show how model (equation 12) fits observed data (100% data fit), and how it might predict what would happen in the future (Figures 13, 14 and 15). The fits shown in the figures are for a specific variant of the model where scaling factor is constant. Variants where it is a linear function of disclosure events or a quadratic functions have been considered in the past for non-security software faults ([7, 16]). We have explored different scaling function shapes and we have found that the predictive power of the model remains roughly the same.

So all this shows that the model (in its present form) is at least consistent with the observed data. However, the reality is that a classical reliability model such as Musas could also fit the disclosure data quite well [2]. The power of the Bayesian model will start showing when we add to the mix the problem repair time delays, and the subjective views that modify the Gamma distribution shape and scaling parameters. As already noted, the current model probably applies to voluntary class of security faults It could also be applied to closed source software for which commercial vendors distribute a fix along with the disclosure. In the cases considered here, we know that all the reported problems were eventually fixed. Also Gamma's α is greater than one. All this indicates that there is a genuine reduction in the security issues. This should increase the trust in this particular open source software.

We plan to extend the model to incorporate more subjective views. For example, delay in problem repair should result in a partial loss of trust, an increase in the perceived probability that the system might fail due to a security

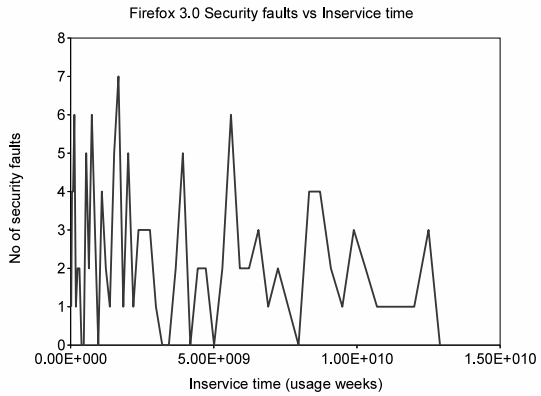

Figure 4: Firefox 3.0 - Security Faults

breach. That probability also will depend on the probability that the system would be attacked (or have its operational profile changed externally). Similarly, a consistently level λ might, after a certain period of time erode confidence in the product despite disclosures that come with fixes. One would really want to see genuinely reducing λ. Of course, as was shown in [4] λ could reduce simply due to advent of a newer version of the product and shifting of end-user interests to that version.

4 Related Work

4.1 Security Models

Alhazmi et al [1] proposed the vulnerability discovery model to estimate the cumulative number of vulnerabilities discovered. The authors base their assumptions on the rate of discovering vulnerabilities by testers and malicious users, but test their model on the vulnerability report data [18]. The vulnerability report data only tells the time vulnerabilities were publicly disclosed and not when they were actually discovered. In their model, they assume that

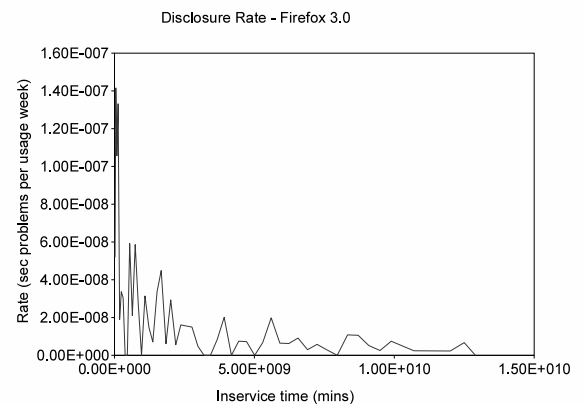

Figure 5: Firefox 3.0 - Disclosure rate

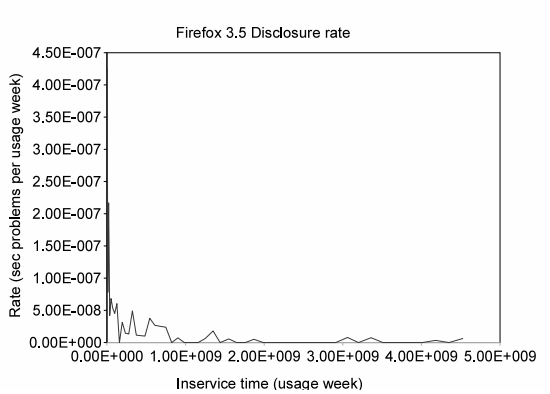

Figure 7: Firefox 3.0 - Disclosure rate

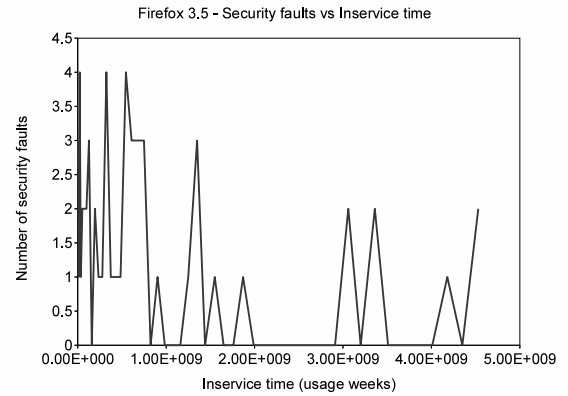

Figure 6: Firefox 3.5 - Security Faults

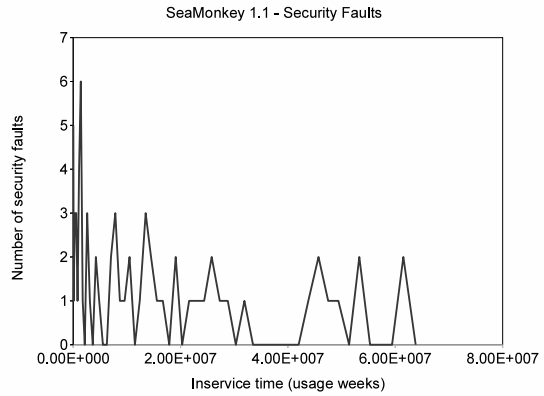

Figure 8: SeaMonkey 1.1 - Security Faults

the number of vulnerabilities (say N) that will eventually be detected for a particular release is finite. Schryen et al [22] showed that their estimated value of N for Windows XP, Windows NT, Redhat 6.2 and Redhat 7.1 did not match with the actual number of problems eventually discovered for the projects. In this paper, we have presented an infinite category model that focuses on the reporting of or disclosure of unique security faults in open source projects.

Anderson [5] proposed a theoretical model based on thermodynamics for discovery and resolution of vulnerabilities. Anderson did not evaluate his model on any real data. Alhazmi et al [1] tested Anderson's model on Win95, WinXP, or Redhat security data and observed that the model does not fit any of the data.

4.2 Application of Classical Reliability Models

When applying classical software reliability models directly on security data, security researchers often forget to justify the assumptions of the reliability models [18]. Rescorla [19] assumed that security faults are similar in nature to non-security faults and follow a reliability growth pattern as that of non-security faults. But this is is not the case for security faults [2]. In [2], we showed that security faults have a characteristic different from that of non-security faults in terms of reporting and correction. Rescorla used the Goel-Okumoto reliability model on security data for Redhat 6.2, WinNT4, Solaris 2.5.1 and FreeBSD 4.0 and was able to fit the model only for Redhat 6.2, and not the others.

Ozment [17] applied seven reliability models to OpenBSD security data set. Ozment classified the dataset under two perspective: failure and fault perspective. He observed that sometimes a set of vulnerabilities would be detected around the same time and a single patch would fix all of them. He termed these as related security faults. Failure perspective data would include the related security faults grouped as one under the notion that one failure could have revealed all those related faults. Fault perspective is where all faults are considered as individual data points. He found that none of the seven reliability models worked on the fault data, while three models with Musa's Logarithmic model being a best fit worked for the failure data. there is no evidence to support that a single failure revealed all the related faults. Further, it is possible to find faults reported around the same time by different sources and different fixes may

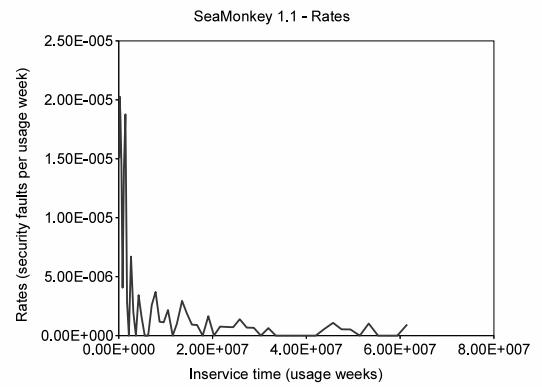

Figure 9: SeaMonkey 1.1 - Disclosure rate

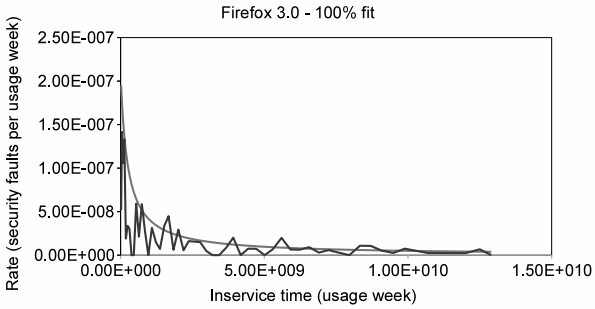

Figure 10: Firefox 3.0 - 100% Fit

be bundled together as a single patch release. Thus grouping bugs as failure perspective seems to be weak since this lacks evidence and excludes data points. Not excluding these data points turns the result otherwise which is the analysis from fault perspective. Nevertheless the author concludes that although he could observe a fit on the data, there is no additional information available to justify the assumptions of the model for security data. Thus the results of this analysis is inconclusive.

5 Conclusion

In this paper, we have presented and discussed a model that provides both an objective and a subjective view of the disclosure of security faults. We have successfully extended the Bayesian approach, used in classical software reliability models for non-security failures, to the security space in describing the disclosure of unique security faults in open source projects. We have assessed our model using three popular open source project releases and identified future direction required to reflect end-user perceptions associated with security problems.

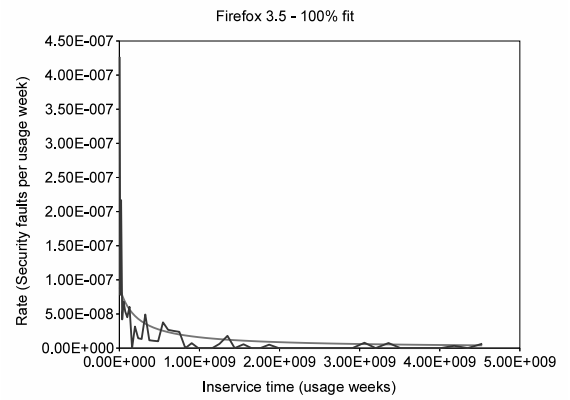

Figure 11: Firefox 3.5 - 100% Fit

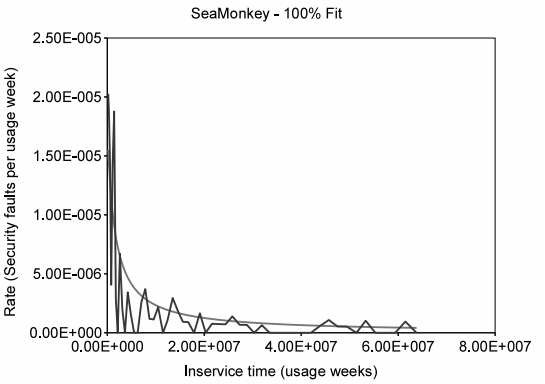

Figure 12: SeaMonkey 1.1 - 100% Fit

References

[1] O. H. Alhazmi, Y. K. Malaiya, and I. Ray. Measuring, analyzing and predicting security vulnerabilities in software systems. *Computers & Security*, 26(3):219–228, 2007.

[2] P. Anbalagan and M. Vouk. Student paper:on reliability analysis of open source software-fedora. In *ISSRE '08: Proceedings of the 19th IEEE International Symposium on Software Reliability Engineering*, Seattle, WA, USA, 2008. IEEE Computer Society.

[3] P. Anbalagan and M. Vouk. On mining data across software repositories. In *Proceedings of the Sixth IEEE Working Conference on Mining Software Repositories*, May 2009.

[4] P. Anbalagan and M. Vouk. Towards a unifying approach in understanding security problems. In *ISSRE '09: Proceedings of the 20th IEEE International Symposium on Software Reliability Engineering*, Bengaluru, India, 2009. IEEE Computer Society.

[5] R. Anderson. Security in open versus closed systems the dance of boltzmann, coase and moore. In *In Conference on Open Source Software Economics*, pages 1–15. MIT Press, 2002.

[6] M. Cox. *Risk report: Four years of Red Hat Enterprise Linux 4, http://magazine.redhat.com/category/red-hat-enterprise-linux/*. Red Hat Magazine, RedHat, 2009.

[7] W. Farr. Software reliability modeling survey. In M. R. Lyu, editor, *Handbook of software reliability and system relia-*

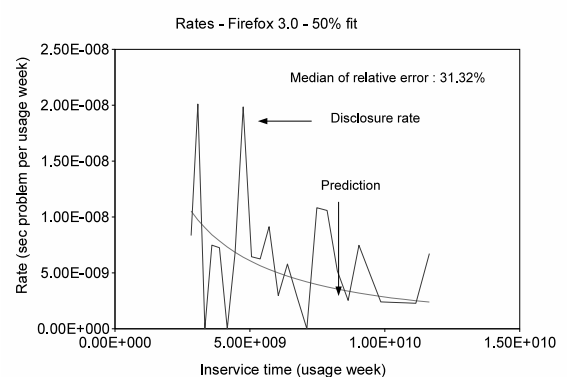

Figure 13: Firefox 3.0 - Prediction

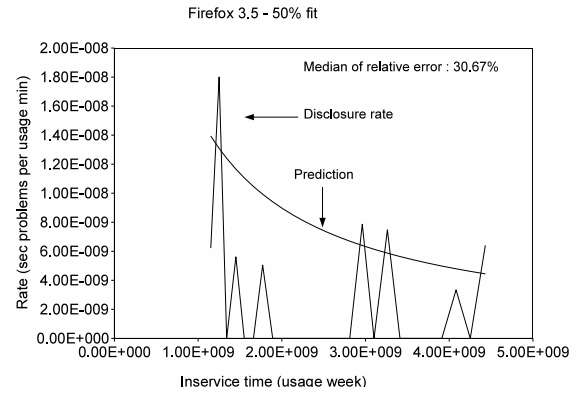

Figure 14: Firefox 3.5 - Prediction

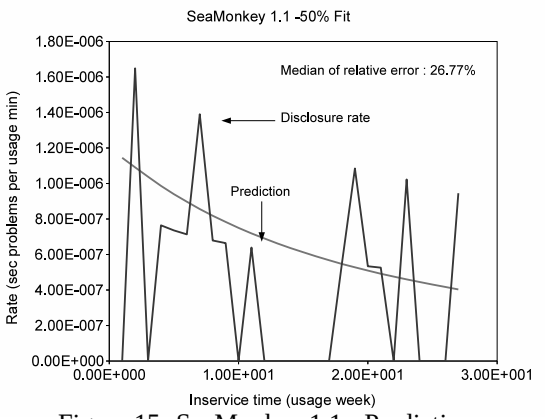

Figure 15: SeaMonkey 1.1 - Prediction

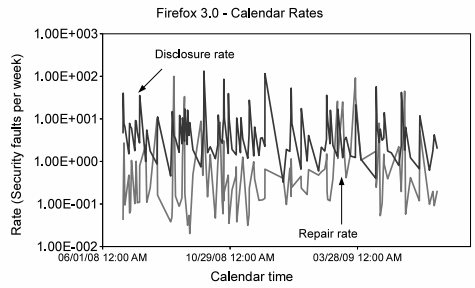

Figure 16: Firefox 3.0 - Calendar Rates

bility, chapter 3, pages 71–117. McGraw-Hill, Inc., Hightstown, NJ, USA, 1996.

[8] W. Jones and M. A.Vouk. Software reliability field data analysis. In M. R. Lyu, editor, *Handbook of software reliability and system reliability*, chapter 11, pages 439–489. McGraw-Hill, Inc., Hightstown, NJ, USA, 1996.

[9] E. Jonsson and T. Olovsson. A quantitative model of the security intrusion process based on attacker behavior. *IEEE Trans. Softw. Eng.*, 23(4):235–245, 1997.

[10] B. Littlewood. How to measure software reliability, and how not to. In *ICSE*, pages 37–45, 1978.

[11] B. Littlewood. Theories of software reliability: How good are they and how can they be improved? *IEEE Trans. Software Eng.*, 6(5):489–500, 1980.

[12] B. Littlewood. A bayesian differential debugging model for software reliability. In *Proceedings of the 1981 ACM workshop/symposium on Measurement and evaluation of software quality*, pages 129–130, New York, NY, USA, 1981. ACM.

[13] B. Littlewood, S. Brocklehurst, N. Fenton, P. Mellor, D. Wright, J. Dobson, J. Mcdermid, D. Gollmann, and E. T. Ex. Towards operational measures of computer security. *Journal of Computer Security*, 2:211–229, 1993.

[14] B. Littlewood and J. L. Verrall. A bayesian reliability growth model for computer software. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 22, No.3:332–346, 1973.

[15] Microsoft Security Response Center. *Responsible Disclosure Policy and Acknowledgement Policy for Security Bulletins, http://www.microsoft.com/security/msrc/report/disclosure.aspx*. Microsoft Corporation, 2009.

[16] J. D. Musa, A. Iannino, and K. Okumoto. *Software reliability: measurement, prediction, application*. McGraw-Hill, Inc., New York, NY, USA, 1987.

[17] A. Ozment. Software security growth modeling: Examining vulnerabilities with reliability growth models. In *In Dieter Gollmann, Fabio Massacci, and Artsiom Yautsiukhin, eds., Quality Of Protection: Security Measurements and Metrics*, May 2006.

[18] A. Ozment. Improving vulnerability discovery models. In *Proceedings of the 2007 ACM workshop on Quality of protection*, pages 6–11, 2007.

[19] E. Rescorla. Is finding security holes a good idea? In *IEEE Security and Privacy*, Jan-Feb 2005.

[20] S. M. Ross. *Introduction to Probability Models, Ninth Edition*. Academic Press, Inc., Orlando, FL, USA, 2006.

[21] N. F. Schneidewind. Reliability - security model. In *ICECCS '06: Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems*, pages 279–288, Washington, DC, USA, 2006. IEEE Computer Society.

[22] G. Schryen and R. Kadura. Open source vs. closed source software: towards measuring security. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 2016–2023, New York, NY, USA, 2009. ACM.

[23] P. Tobias. *Assessing Product Reliability, Handbook of Statistical Methods*. National Institute of Standards and Technology, and SEMATECH, Washington, DC, USA, 1991.