The Likelihood of Vulnerability Rediscovery and the Social Utility of Vulnerability Hunting

Andy Ozment[*]
Computer Laboratory, University of Cambridge

Note The trend analysis described in section 4 has been updated in more recent work that uses a larger data set:

> Andy Ozment and Stuart E. Schechter. "Milk or Wine: Does Software Security Improve with Age?" In the proceedings of *The Fifteenth Usenix Security Symposium.* July 31 - August 4 2006: Vancouver, BC, Canada.

The updated analysis shows that the depletion of vulnerabilities starts later in the lifecycle of OpenBSD and is less rapid than predicted in section 4. The updated analysis should thus be used in place of the results in section 4. Nonetheless, the overall conclusion, that the rate of vulnerability reporting in the legacy code base of OpenBSD is declining over time, remains the same.

The data on vulnerability rediscovery presented in section 5 remains pertinent and is unique to this paper. This data is used to refute the assertion, made elsewhere, that vulnerability hunters are unlikely to discover the same vulnerability.

Abstract

Initial attempts to apply software reliability growth models to the process of vulnerability finding relied upon noisy data. Here, a more appropriate data collection process is discussed and employed to identify the age of vulnerabilities in OpenBSD 2.2. A number of models are tested against the data and two are found to have acceptable goodness-of-fit. These models indicate that the pool of vulnerabilities in the system is being depleted. However, models that fit the data but do not indicate depletion may also exist. While this result is thus not conclusive, it does suggest that more investigation is needed and that, contrary to prior work, vulnerability depletion cannot yet be ruled out. It is thus possible that vulnerability hunting can result in a more secure product and can provide a social benefit. Patch announcements and vulnerability reports are also used to quantitatively (albeit roughly) demonstrate that vulnerabilities are often independently rediscovered within a relatively short time span. This finding provides a quantitative and qualitative rationale for vulnerability disclosure policies intended to pressure vendors into more rapidly providing patches. Furthermore, information on the likelihood of rediscovery can be added to these models in order to more accurately establish appropriate disclosure windows—and when disclosure may not be optimal at all. Although neither result is conclusive, both contradict previous work by providing support for the conclusion that vulnerability hunting is socially useful.

[*]This work was funded by a Marshall Scholarship and a UK ORS Award.

Version 1.1. Version 1.0 was presented at:
The Workshop on Economics and Information Security. June 2-3, 2005: Cambridge, MA, USA.

1 Introduction

There exists a significant community of individuals who are interested in computer security and who take it upon themselves to discover and report software vulnerabilities. These 'benign identifiers' are not necessarily employed by or associated with the vendor of the products they analyze; nor are they attackers, those whose primary aim is to use the vulnerabilities to unlawfully attack systems.

Nonetheless, the means by which the benign identifier informs the vendor and the public of a vulnerability is a topic of dispute. In the past, a benign identifier might have notified the vendor of the vulnerability and then waited an indeterminate length of time until the vendor released a patch that fixed it. In the 1990s, discontent with the length of time some vendors were taking to release patches led to the creation of public 'full-disclosure' fora (e.g. the Full Disclosure and Bugtraq mailing lists). In these fora, benign identifiers publish vulnerability reports that detail the existence of the vulnerability and sometimes go so far as to give instructions on its exploitation.

The creation of the full-disclosure fora provoked a policy debate: was public welfare served better by releasing a vulnerability report before the vendor's patch (*instantaneous disclosure*) or from delaying the report until the patch was ready (*responsible disclosure*)?[1] Advocates of instantaneous disclosure argue that it enables users to mitigate the impact of the vulnerability and pressures software vendors to provide patches more rapidly. Advocates of responsible disclosure argue that notifying attackers of the existence of the vulnerability without simultaneously providing a patch results in increased social cost due to attacks. Implicit in both arguments is the assumption that vulnerabilities should be found and fixed because they are likely to be rediscovered by malicious actors: if a vulnerability has been found by a benign identifier, it should be fixed before an attacker independently rediscovers and exploits it. In [Res04], Rescorla questions this underlying assumption and argues persuasively that:

1. Vulnerability hunting does not significantly decrease a product's pool of latent vulnerabilities during the product's life span. It thus does not result in a significantly higher quality or more secure product.

2. The independent rediscovery of a vulnerability is unlikely: a benign identifier is unlikely to identify vulnerabilities that would otherwise be discovered by an attacker.

3. As a result, both instantaneous and responsible disclosure provide little benefit. Indeed, from a public welfare perspective, they are actually detrimental: vendor patches and vulnerability reports by benign identifiers

[1] Nomenclature like 'benign identifier' and 'responsible disclosure' is not value free and is hotly debated. I use it out of convenience and in accordance with its usage in the popular media. When possible, I employ the terminology of [ATX04] and [KT04] to facilitate the creation of a standard nomenclature.

alert attackers to vulnerabilities they would not have discovered on their own.

However, Rescorla notes that his data set is noisy and problematic, and he recommends the collection of better data. This work answers his challenge by using a better data set to re-examine his arguments and also by considering new evidence from deployed software. First, the stochastic software reliability models used in the initial work are considered and it is shown that the data set does not adequately conform to the requirements of these models. The creation of a data set that does comply with these requirements is discussed; such a data set is then analyzed. The results are not definitive, but they suggest that vulnerability hunting may deplete the pool of latent vulnerabilities in a product. As a result, such efforts may provide the social benefit of a more secure product. Second, alternative data sources are used to demonstrate in a less mathematically rigorous manner that some non-trivial level of rediscovery is in fact occurring with respect to Microsoft products.

These results suggest that the social utility of vulnerability hunting is more complicated than previously argued, and that it may actually be socially beneficial. Thus, where prior work could assume that some inputs to a social welfare model were negligible or overwhelming, the findings here suggest that a more careful analysis of the cost and benefits of vulnerability hunting is required.

2 Relevant Literature

The relevant literature can be divided into two groups. Section 2.1 provides an overview of the policy debate between advocates of responsible disclosure and advocates of instantaneous disclosure. Section 2.2 presents work that challenges the assumptions underlying the disclosure policy debate and briefly discusses the literature on software reliability growth models.

2.1 Vulnerability Disclosure Policy

Advocates of responsible disclosure do acknowledge that vendors may not release patches promptly or at all without the benign identifier threatening to make the vulnerability public; many solve this dilemma by giving vendors a deadline. Three prominent responsible disclosure policies emphasize timely and continuing contact between the benign identifier, the coordinating center (e.g. CERT/CC[2]) and the vendor in order to ensure that the patch is released promptly: [CER00], [Pup01], and [CW02].[3] The CERT/CC policy is perhaps the most important, as CERT/CC is often used as a mediator between benign identifiers and vendors. Their policy gives a forty-five day deadline for the vendor to resolve the problem, although extensions may be granted if the vulnerability is a particularly difficult one to remediate. The latter two policies emphasize that the reporter and vendor

[2]CERT/CC was previously the Computer Emergency Response Team Coordinating Center

[3]The [CW02] policy has been retracted by its authors.

should negotiate to ensure that the vendor has a reasonable length of time to create a patch while not unnecessarily dragging its feet.

Arora *et al* empirically compare the results of the instantaneous disclosure policy with those of the responsible disclosure policy. They find that instantaneous disclosure does force vendors to provide patches more rapidly than otherwise, and that it also increases the probability that the vendor will provide a patch *at all*. They note, of course, that the number of attacks against that vulnerability is increased (possibly from a starting point of zero) by instantaneous disclosure and thus that overall social utility may decline [AKN+04].

In a separate work, Arora *et al* model a policy for disclosure in which social planners (e.g. CERT/CC) threaten to publicly disclose a vulnerability if the vendor does not provide a patch before a deadline. Vendors are averse to bearing the increased cost of developing a patch rapidly rather than more leisurely. In Arora *et al*'s model, the social planner is explicitly motivated by the fear of *attacker independent rediscovery*: that a malicious actor will independently rediscover the vulnerability and then exploit it. Their model includes the time that an attacker independently rediscovers the vulnerability as stochastic, with a likelihood that increases as time passes since the vulnerability was found by the benign identifier.

They find that neither instantaneous disclosure nor non-disclosure is socially optimal. Furthermore, without a deadline, vendors do not provide patches rapidly enough. In their model, the social planner discloses after a period less than that desired by the vendor, so that vendors are motivated to release patches more quickly. Early disclosure (if it does not result in a hurried patch release by the vendor) thus accepts some losses due to exploitation against a longer delay in patch release, which may have resulted in attackers independently rediscovering the vulnerability. Their results are not altered when they expand the model to include incomplete compliance in users' implementing patches [ATX04]. Unfortunately, they fail to weigh the costs of early disclosure against the risk of attacker independent rediscovery.

In a later work, Hasan Cavusoglu, Huseyin Cavusoglu, and Srinivasan Raghunathan also create a game theoretic model of the vulnerability disclosure process [CCR05]. They consider the amount of time venders are given to create a patch before the vulnerability information is made public—including the extremes of instantaneous public disclosure or never disclosing the vulnerability. They model the benefits and costs of vulnerability disclosure policies to vendors, social coordinators (e.g. CERT/CC), vulnerability identifiers, and users. They find that no single policy is always optimal: instead, the social coordinator should alter the amount of time available to the vendor in accordance with the cost of creating a patch (e.g. its difficulty), the vulnerability's risk, and the user population. Although no single policy is optimal, an optimal policy does always exist. However, for vulnerabilities that affect multiple vendors, no policy can guarantee that all vendors will release a patch.

In both [ATX04] and [CCR05], the impetus for releasing patches quickly is the fear of attacker independent rediscovery. Although Rescorla argues that the likelihood of such rediscovery is small, the results below demonstrate that it is non-negligible and should not lightly be dismissed.

2.2 Security Reliability Modeling

One of the key assumptions underlying the conclusion of [ATX04] is the belief that attackers are likely to rediscover any vulnerability found by a benign identifier; however, Rescorla argues that vulnerability discovery is a largely stochastic process and the likelihood of rediscovery is low [Res04]. In that work, Rescorla gathered vulnerability finding information from the ICAT vulnerability database and examined it in two ways: from the perspective of products and from the perspective of vulnerabilities. For the former, a single version of four operating systems was selected,[4] and the number of vulnerabilities found per quarter in each operating system was analyzed. For the latter, he examines the age distribution of 1,391 vulnerabilities found between 1997 and 2002 (not just those vulnerabilities related to the four operating systems).

Rescorla's analysis is based upon software reliability growth modeling (see [Goe85], [AIA93], and [Lyu96] for summaries of the literature). This field has produced a number of different models of the process by which faults are identified. (In this paper, I use *fault* interchangeably with *bug*: both refer to a software design or implementation flaw that is not related to security. Flaws related to security are referred to as *vulnerabilities*.) Typically, a software engineer will plot the identification of faults versus time and then attempt to fit a stochastic model to the data points. Models that fit within acceptable confidence intervals are then used to determine: the total number of faults in the system, the amount of testing remaining until the next fault is identified, the amount of testing necessary to identify some percentage of the total faults, etc. Reliability growth models can be divided into those which require the data to be the time-between-faults and those which require the number of faults found in a set interval of time (i.e. failure count models). The former models may require that program execution time be tracked, while the latter are often used with calendar time. These models thus test whether the *reliability* of the program is increasing: when the models fit the data, they are essentially modeling the decrease in the pool of flaws that exist in the product and the corresponding decrease in the likelihood that a flaw will be discovered. When such tests are performed with respect to vulnerabilities, they are used to test whether the *security* of the program is increasing: e.g. that the number of vulnerabilities remaining in the program is significantly decreasing.

Rescorla applies both a linear and an exponential Goel-Okumoto [GO79] model to his program-perspective data. Neither model provides an acceptable fit and thus neither suggest that the security of these programs is increasing. For the vulnerability-perspective data, vulnerabilities were divided into four cohorts based on year of introduction (1997–2000). Only the 1999 cohort shows any significant trend towards a decrease in the number of remaining vulnerabilities.

[4]Windows NT 4.0 (released 1996-08), Solaris 2.5.1 (1996-05), FreeBSD 4.0 (2000-03), and Redhat Linux 7.0 (2000-08). The ICAT database Rescorla utilized was from 2003-05-19.

Using the entire data set (rather than single-year cohorts), both an exponential and a Weibull distribution provide reasonable fits; however, even the more optimistic of these shows the half-life of a vulnerability to be approximately 2.5 years.

Rescorla thus argues that the pool of vulnerabilities in a product is not significantly diminished during that product's life span; as a result, he concludes that the pool of vulnerabilities in a product is essentially infinite with respect to the lifespan of the product. Therefore, an attacker is unlikely to independently rediscover a vulnerability that has been previously discovered by a benign identifier: the pool of vulnerabilities is too large (and he assumes that vulnerability finding is stochastic).[5] Thus, while Arora *et al* conclude that a social planner should impose a deadline by which vendors must release a patch, Rescorla concludes that full disclosure (both responsible and instantaneous) is not socially beneficial at all [Res04].

To reach this conclusion, Rescorla models the costs suffered by users after full disclosure, either benign or instantaneous. He first notes that users patch their systems slowly (even when it is known that a vulnerability is being exploited), and some may not patch at all [Res03]. Furthermore, a vulnerability report or a patch will inform attackers of a vulnerability and help them to create an exploit for that vulnerability. By forcing vendors to release patches, benign identifiers ensure that attackers will also be aware of the vulnerability. Since many systems remain unpatched for some time after the patch is released, those users will likely suffer the effects of attacks—although unpatched systems may be those least valued. Rescorla also questions the argument that benign identifiers should seek to find vulnerabilities because those vulnerabilities may *already* have been found and utilized by attackers. He believes that an attacker cannot exploit a vulnerability many times before the vendor learns about the vulnerability: the attacker will eventually attack a system that is being carefully watched, thus alerting the system administrators to the vulnerability. Using these assumptions, Rescorla models the social utility of vulnerability hunting (the work performed by benign identifiers) and concludes that it is not socially beneficial.

3 Data Sets

Rescorla's program-perspective results indicate no significant decrease in the number of vulnerabilities remaining in the four programs he examined, and the vulnerability-perspective results indicate a vulnerability half-life of at best 2.5 years. However, the ICAT data set [NIS03] with which he worked is problematic in a number of ways. Section 3.1 discusses the data requirements of software reliability growth models, while Section 3.2 notes the areas in which the ICAT

[5]Rescorla does note that vulnerability rediscovery happens on occasion and thus concludes that some vulnerability hunting is not stochastic. However, he finds this situation rare enough to exclude it from the cost-benefit analysis. Section 5, below, provides evidence that independent rediscovery occurs too frequently to be dismissed and discusses this phenomenon in more depth.

data set fails to fulfill the model requirements. The data collection technique used in this work was selected to overcome the failures of the ICAT data set; it is discussed in Section 3.3.

3.1 Model Requirements

While the various models have different applicability requirements, the effectiveness of *all* software reliability models is dependent upon three key assumptions [Goe85] [Lyu96]. They require:

1. **A valid operational profile:** 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.

2. **Static software:** Significant alterations cannot be made to the software being tested. In general, the only alterations permitted are those necessary to repair faults. Although models like the "Goel Okumoto imperfect debugging model" can cope with fault patches introducing new faults, no model can adequately cope with a significant amount of code 'churn' due to added features.

3. **That time is normalized for effort:** The models' concept of time should encompass effort. If program execution time is utilized, this assumption is clearly satisfied. However, if calendar time is used then it must be normalized for the number of testers participating [Wal01].

From a narrow perspective, Assumption 1 casts doubt upon the applicability of software reliability models to vulnerabilities: many vulnerabilities rely upon the adversary intentionally inputting abnormal data (that is, data outside of the bounds of a normal operational profile). However, a more broad perspective is that the operational profile includes *all* possible input. Although this perspective indicates that software reliability models can be applied to vulnerabilities, it does imply that vulnerabilities may be identified more slowly than bugs would be identified.

In addition, chronologically local 'spikes' in the number of vulnerabilities found may indicate the discovery of a new technique for identifying vulnerabilities. The discovery of a new technique is conceptually equivalent to an expansion of the operational profile. It also violates an additional assumption that underlies all of the stochastic reliability growth models: that the times between failures are independent. Interval models introduce some degree of data smoothing and thus can better compensate for violations of this assumption than time-between-faults models.

3.2 ICAT Dataset Shortcomings

The ICAT database used by Rescorla has a number of shortcomings, all but one of which are corrected by the data collection technique employed in this paper.

The ICAT shortcomings are chronological inconsistency, incomplete selection, and lack of effort normalization.

The ICAT database has inadequate information on the important chronological information of vulnerabilities: their *birth date* (i.e. when the vulnerable code was added to the program) and their *death date* (i.e. when they were identified). The birth date of vulnerabilities may be inaccurate because vulnerabilities may predate those versions of a program for which they are listed in ICAT: the reporter of the vulnerability frequently tests it against only the most current version of a program. However, this shortcoming does create a bias in favor of vulnerability depletion.

The death date of a vulnerability is also only approximately recorded. Before 2001, ICAT reports a "published before" date; after 2001, it reports the date when the vulnerability was added to the database. Both approaches may be inaccurate by several months. Furthermore, even if ICAT does report an accurate date, that date may be when the vulnerability became public. Some vendors work on vulnerabilities for months or even years before they release a patch and the vulnerability becomes public.

Another drawback of the ICAT database is that it does not contain every vulnerability identified or even a consistent subset of vulnerabilities. ICAT contains only those vulnerabilities that have been assigned CVE identifiers; many vulnerabilities that predated the creation of the CVE database in September 1999 have never been assigned CVE identifiers.[6] Even after that date, not all vulnerabilities recorded by the vendor or other databases like Bugtraq are included.

Finally, Rescorla notes that Assumption 3 is also violated: the data is not normalized for the number of testers (both malicious and benign identifiers). For vulnerability identification, of course, the number of testers does not necessarily equate to the number of users. Only a subset of the user population will be knowledgeable enough to identify and report vulnerabilities. Furthermore, this subset is also not necessarily a fixed proportion of the total user population: because these knowledgeable testers may be interested in programs popular in their community or currently prominent in the media, the ratio of testers to users may differ between programs and also across time.

3.3 Collection Technique for the OpenBSD 2.2 Dataset

The OpenBSD operating system was selected as a data source favorable to the hypothesis that vulnerability hunting can result in a significant decrease in a program's vulnerabilities. The OpenBSD security philosophy is in favor of vulnerability hunting and full disclosure:

> "OpenBSD believes in strong security. Our aspiration is to be NUMBER ONE in the industry for security (if we are not already there)....

[6]CVE, the Common Vulnerability and Exposure database is an initiative to provide a single, widely accepted name to vulnerabilities. Rather than a database of information about a vulnerability, it simply provides a universal identifier for that vulnerability. ICAT is a database that provides information on CVE identified vulnerabilities [MIT05].

> Like many readers of the BUGTRAQ mailing list, we believe in full disclosure of security problems. In the operating system arena, we were probably the first to embrace the concept. Many vendors, even of free software, still try to hide issues from their users. Security information moves very fast in cracker circles. On the other hand, our experience is that coding and releasing of proper security fixes typically requires about an hour of work – very fast fix turnaround is possible. Thus we think that full disclosure helps the people who really care about security." [Ope05]

The earliest stable version of OpenBSD for which vulnerability information is available is 2.2, released on December 1, 1997. Vulnerability information prior to 2.2 is not accurate because the developers performed an extensive source code audit and silently (i.e. without announcing them publicly) fixed a number of vulnerabilities. Version 2.2 was thus selected for study.

The collection technique utilized overcomes the chronological inconsistencies that plague the ICAT database. OpenBSD makes every version of its source code available via an internet accessible CVS repository (version control system). This repository allows the examination of every change and addition made to the source code. For each vulnerability, the source code was examined to find its death date: the date that the vulnerability was repaired in the code. Because the date utilized is the date on which the repair was checked into the CVE repository, it closely tracks the date on which OpenBSD was informed. As a result, even vulnerabilities that impact multiple vendors and thus must be kept silent for long periods could be accurately tracked.

The vulnerability was then traced back through every preceding change to the relevant code. The date on which the vulnerable code was first written was recorded as its birth date. 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.

A wider selection of vulnerability databases was also used, in order to overcome the coverage problems inherent to the ICAT database. A vulnerability was included if it was cited as existing by either CVE (i.e. ICAT) or the Bugtraq vulnerability database [Sec05]. In addition, any vulnerability for which OpenBSD provided a patch was also included [Ope05]. The use of Bugtraq and OpenBSD patch announcements compensate for the shortcomings of ICAT; in particular, the OpenBSD patch announcements provide coverage prior to the existence of the other two databases. The initial dataset included any vulnerability from these sources that was made public between 1997-12-01 and 2000-05-31. This 30 month period included all of the vulnerabilities listed for five releases of the operating system (versions 2.2—2.6).

Unfortunately, this work does not present a solution to Assumption 3: the OpenBSD 2.2 chronological data is not normalized with respect to effort. OpenBSD does not track usage patterns; because it is often used as a server operating system, other available sources of usage data are also inadequate (e.g. the number

Program Version	Date Released	Vulns Introduced*	Total Vulns**
OpenBSD 2.2	1997-12-01	39	39
OpenBSD 2.3	1998-05-19	2	30
OpenBSD 2.4	1998-12-01	1	22
OpenBSD 2.5	1999-05-19	1	13
OpenBSD 2.6	1999-12-01	0	4

*The number of vulnerabilities that were born during the period between the previous version and this one. For the purpose of this study, no version existed prior to 2.2. **The total vulnerabilities listed for each version are those that existed in the code prior to that version's release date and were identified after that version's release date.

Table 1: Vulnerabilities Identified 12/97 – 05/00

of web page requests that originate from an OpenBSD system). Section 6 discusses a potential research approach to solving this problem.

Of the initial 44 vulnerabilities identified, 39 dated to OpenBSD 2.2 (see Table 1). The other four occurred in code that was more recent than that release, and one proved not to be a vulnerability. The 39 vulnerabilities were the data source used for the reliability modeling. Note that the data in Table 1 does not necessarily indicate that OpenBSD 2.6 has fewer vulnerabilities overall than OpenBSD 2.2. Rather, it is potentially an artifact of the right censoring of the study: with respect to the study's time frame, code introduced in OpenBSD 2.6 was accessible to testers for far less time than that of OpenBSD 2.2.

Some vulnerabilities shared the same death date or had death dates that were chronologically close. These data spikes may be caused by vulnerability dependencies: e.g. publicity for one vulnerability causes other individuals to search in the same area [Goe85]. This scenario violates the assumption of a stochastic distribution; however, any bias it introduces is against the hypothesis that security is increasing.

4 Security Growth in OpenBSD 2.2

Two parametric software reliability models support the hypothesis that vulnerability discovery can increase the security of a product. Both *Brooke's & Motley's Discrete SR Binomial Model* and *Yamada's S-Shaped Reliability Growth Model* [YOO83] indicate that the vulnerabilities in OpenBSD 2.2 are being depleted. The former model incorporates the possibility that faults will be introduced into the code through the patching process. The latter model describes an S-shaped curve that assumes an initial learning process.

Table 2 lists the ten interval models that were applied to OpenBSD 2.2 vulnerability counts: 30 intervals of one month's length. The standard deviation of the data was 1.64, and the variance was 2.70. The B&M Binomial and Yamada S-Shaped models had acceptable chi square goodness-of-fit results, using maximum likelihood parameter estimation.[7] The results of these tests are shown in Table 2, with the B&M Binomial and Yamada S-Shaped models highlighted.

[7]The SMERFS^3 reliability engineering tool was used to assess the models' applicability [Sto03].

Model	Accuracy	Chi Square*	χ^2 DOF
1. Brooke's & Motley's Discrete SR (Binomial)	23.27	9.42	4
2. Brooke's & Motley's Discrete SR (Poisson)	22.94	9.72	4
3. Gen. Poisson Weighting 1 (Schick-Wolverton)	N/A	9.72	4
4. Gen. Poisson Weighting 2 (Alpha Input)	0.00	9.72	4
5. Gen. Poisson Weighting 3 (Alpha Estimated)	N/A	0.00	0
6. Non-homogenous Poisson Model for Intervals	0.00	10.83	4
7. Schneidewind Model (Treatment Type 1)	0.00	10.83	4
8. Schneidewind Model (Treatment Type 2)	N/A	2.56	0
9. Schneidewind Model (Treatment Type 3)	N/A	83.12	3
10. Yamada's S-Shaped Reliability Growth	22.95	5.58	4

*The Chi Square tests were run with the default cell combination frequency of 5.

Table 2: Interval Model Results for 30 Single-Month Intervals

Probability of Detecting Faults	0.041
Total No. of Faults	49.63
Total No. of Faults Remaining	10.63

Table 3: Brooke's & Motley's Discrete SR Model (Binomial) Results

The accuracy column lists the prequential likelihood value, used to differentiate between the predictive accuracy of competing models [AGCL86]; the B&M Binomial and Yamada S-Shaped model were not significantly different in this regard. Seven time-between-faults models were also tested; as expected, they were not appropriate for the data set—presumably due to its lack of granularity [PTL93].

The fault predictions of the B&M Binomial model are listed in Table 3. Using a maximum likelihood test, the 'probability of detecting faults' parameter best fits the data at a value of 0.041. The B&M Binomial model assumes that this parameter is constant for all test runs and is independent of detecting flaws [AIA93]. The B&M Binomial model also assumes that fixes introduce new flaws at a constant rate. Here, the reintroduction probability parameter is estimated as 0.15 faults introduced per fix. Using these parameters, the model supports the hypothesis of increasing security. The model estimates that 10.63 faults remain in OpenBSD 2.2. (The total number of faults remaining is an expected value and thus need not be an integer.) Appendix A has more information on the model.

The fault predictions of the Yamada S-Shaped Reliability Growth Model are listed in Table 4. The Yamada model is a variation of the Nonhomogeneous Poisson Process Goel Okumoto model employed by Rescorla. Yamada *et al* considered the fault detection process to be S-shaped, with an initial learning curve followed by exponential growth in detection and then diminishing returns to testing [Lyu96]. The assumptions of the Yamada model thus seem at odds with the character of OpenBSD 2.2. Most of the source code (and thus the vul-

	Lower 95% CI	Estimate	Upper 95% CI
Proportionality Constant	0.089	0.132	0.175
Total No. of Faults	39.0	43.08	57.31
Total No. of Faults Remaining	0.0	4.08	18.31

Table 4: Yamada's S-Shaped Reliability Growth Model Results

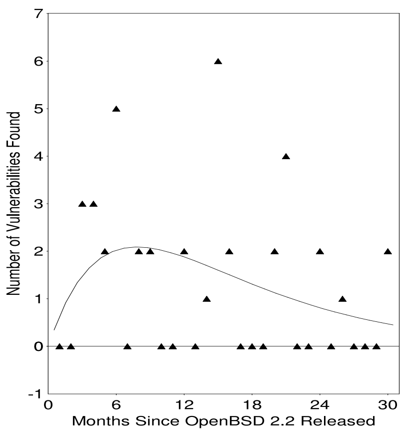
nerabilities) for this release undoubtedly predates it by years: why should there be a peak in vulnerability detection immediately after this release? However, it is possible that OpenBSD's security claims with the release of 2.2 prompted an increase of interest in vulnerability hunting for this operating system. ("We believe that we are NUMBER ONE in the industry at the moment" [Ope98].) The Yamada model estimates that 4.08 faults remain in OpenBSD 2.2; with a 95% confidence interval, between 0 and 18.31 faults are predicted to remain. Appendix B has more information on this model.

The application of multiple reliability growth models to the carefully collected OpenBSD 2.2 data set is thus a partial success. Two fitted parametric models, B&M Binomial and Yamada S-Shaped, support the hypothesis of significantly increasing security. The growth curves of both models are shown with the points from the data set in Figure 1. However, the two predictions for total number of faults do vary more than is desirable. The expected total number of faults remaining is 10.63 for the B&M Binomial model and 4.08 for the Yamada model. Moreover, the upper 95% confidence estimate for the Yamada model is that 18.31 faults remain in OpenBSD 2.2: if the true number of faults is more close to this estimate, then only two-thirds of the vulnerabilities present in OpenBSD 2.2 were discovered in its first thirty months of utilization.

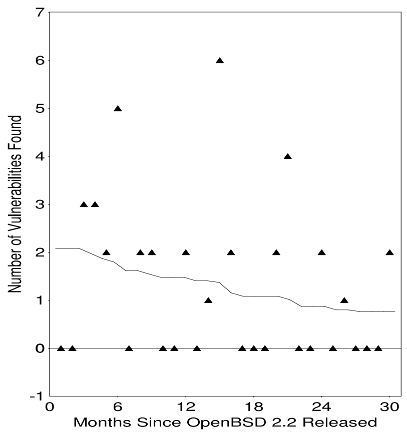
However, identifying two-thirds of the vulnerabilities in OpenBSD 2.2 is more impressive than it might first appear. Even thirty months and five versions later, the vast majority of the OpenBSD code base was created in version 2.2. For example, note that only four vulnerabilities have been identified as having been added between 1997-12 and 2000-05.

Although two of the models test had acceptable goodness-of-fit, this result does not conclusively indicate that vulnerabilities are being depleted. Other models, not indicative of depletion, may also fit the data.[8] In contrast to the results of [Res04], this work has shown a clean data set with which reliability growth models fit. Although the data utilized in this work was significantly more clean, it was also significantly smaller: more data on both this system and others is clearly desirable (see Section 6, Future Work). Vulnerability depletion through vulnerability hunting should not be ruled out, but nor can it be conclusively asserted.

[8]The author thanks Eric Rescorla for pointing out this fact.



(a) Fitted Yamada's S-Shaped Model



(b) Fitted Brooke's & Motley's Discrete SR Binomial Model

Figure 1: Models with Acceptable Fit to the Data Set

5 Examples of Vulnerability Rediscovery

If vulnerability rediscovery is likely, then vulnerability hunting by benign identifiers provides another benefit: vulnerabilities are found and fixed before they would otherwise have been discovered and exploited by attackers. One means of ascertaining the likelihood of vulnerability rediscovery is to search for examples of this phenomenon occurring. A number of examples were found, although the data is not precise enough for a rigorous mathematical analysis. Nonetheless, enough verified examples were found to conclusively disprove Rescorla's assumption that rediscovery is unlikely and can be neglected [Res04, 14].

5.1 Evidence of Independent Rediscovery

Examples of independent rediscovery of vulnerabilities are by their nature hard to find. If the first benign identifier of a vulnerability follows a full-disclosure policy, then it is difficult for a future benign identifier to claim that she identified that vulnerability independently. The best source of such information thus seems to be software vendors who receive more than one report of a vulnerability while they are working on the patch. Unfortunately, not all vendors record the additional reports. However, Microsoft security patch bulletins do at times credit multiple individuals/organizations for reporting vulnerabilities.

As a source of information, these bulletins are very limited. First, the multiple individuals/organizations credited may have collaborated on finding the vulnerability, rather than identifying it independently. Furthermore, the creation of a patch seems usually to require less than three months and only rarely takes more than seven months: the window of time for recording independent

	Number of Reporters				
Year	Not Credited	1	2 Independent	3 Independent	Total
2002	62	71	5	0	138
2003	22	43	4	0	69
2004	22	54	3	2	81
Total	106	168	12	2	288

Table 5: Vulnerability Credits in MS Security Bulletins

rediscoveries is thus fairly short. In addition, benign identifiers credited with independently identifying the same vulnerability may have actually found two reasonably different security flaws that were then linked into one vulnerability. Finally, Microsoft may or may not have a policy to acknowledge multiple independent reporters. Although they have at times done so, they may have omitted the repeated discoveries at other times. Nonetheless, the security bulletins provide a means for ascertaining at least some portion of the times when a vulnerability has been independently reported to Microsoft.

Fortunately, security professionals often release their own vulnerability report to the public on the date that the vendor releases its patch, and that report may be used to ascertain when individuals worked together. For this data, vulnerabilities are credited as having been identified independently if at least one of the two reporters (or, correspondingly, two of the three reporters) asserted via the public vulnerability report or email correspondence with the author that he independently identified the vulnerability.

Table 5 enumerates the individual vulnerabilities announced by Microsoft via security bulletins. In the second column is the number of vulnerabilities for which Microsoft provided no reporting credit: presumably these vulnerabilities were either discovered internally or publicly announced before they were reported to Microsoft (i.e. instantaneous disclosure). The third column reports the number of vulnerabilities for which Microsoft identified exactly one reporter. The fourth column lists the number of those vulnerabilities that had exactly two independent reporters; the fifth column lists the number of those vulnerabilities that had exactly three independent reporters. The final column is total number of vulnerabilities noted by Microsoft in security bulletins.

Table 6 summarizes the total number of confirmed multiple reports in the second column (i.e. those vulnerabilities for which two or three individuals independently reported the vulnerability to Microsoft). The third column is the total number of vulnerabilities for which anybody is credited: both single reports and multiple reports. The final column is the percentage of vulnerabilities for whom multiple individuals are credited with the report. The final column thus represents the percentage of vulnerabilities that were independently rediscovered in the reasonably short time frame that Microsoft was working to create a patch.

Table 7 lists each vulnerability that was rediscovered and notes the date of each independent report (as remembered or recorded by the reporter), the date

Year	Multiple Reports*	All Credited Reports	Multiple/Total
2002	5	76	6.58%
2003	4	47	8.51%
2004	5	59	8.47%
Total	14	182	7.69%

*Reports credited to either two or three independent reporters.

Table 6: Vulnerability Credits in MS Security Bulletins

	Report				
CVE	1st*	2nd	3rd	Date Public	Age**
2002-0018	2000-10-31	?		2002-01-30	456 days
2002-0074	2001-12-03	?		2002-04-10	128 days
2002-0641	2002-05-28	?		2002-07-10	43 days
2002-0693	2002-07-31	?		2002-10-02	63 days
2002-1145	2002-08-23	?		2002-10-16	54 days
2003-0226	c.2003-01-15	?		2003-05-28	133 days
2003-0228	2003-03-14	2003-03-23		2003-05-07	54 days
2003-0528	c.2003-07-23	2003-07-29		2003-09-10	49 days
2003-0662	c.2003-04-15	?		2003-10-15	183 days
2003-0908	c.2003-10-15	?	?	2004-04-13	181 days
2004-0123	c.2004-01-13	?		2004-04-13	91 days
2004-0212	2004-05-06	2004-06-20	2004-07-07	2004-07-13	68 days
2004-0214	c.2002-04-15	2004-08-03		2004-10-12	758 days
2004-0216	c.2003-04-15	2004-07-12		2004-10-12	546 days

*Dates preceded by a 'c.' are approximate. When the reporter remembered only the week or month, the middle of that period was chosen. **The length of time that elapsed between when the vulnerability was first reported to Microsoft and when the patch was released.

Table 7: Timing of Rediscovery

that the vulnerability became public (when the patch was released), and the vulnerability's age (the total length of time between when the vulnerability was first reported and when it was made public through a patch release). If not every date on which the vulnerability was reported to Microsoft is known, then the earliest known date is used: this practice may underreport the age.

These instances of multiple reporting make clear that vulnerability rediscovery is not unknown. The most interesting information would be to look for a relationship between the length of time the vendor works on a patch and the likelihood of rediscovery. Unfortunately, complete information on the dates when vulnerabilities were reported is unavailable: not every reporter could be reached or could remember the date on which she reported the vulnerability.

Nonetheless, the information that these bulletins provide is a clear indicator that vulnerability rediscovery occurs 'in the wild' and that it is not particularly uncommon. If anything, this analysis undercounts them due to the nature of vulnerability disclosure and announcements.

5.2 Possible Reasons for Rediscovery

Intuitively, vulnerabilities are likely to be rediscovered for a number of reasons. First, the use of similar tools to find vulnerabilities is an obvious source of potential rediscoveries. Second, if the key functionality or security critical portion of a product is small, then interested individuals will have only a limited area to search for vulnerabilities. Third, when a new class of vulnerability (e.g. integer overflows) is discovered, individuals will search for instances of that vulnerability in the products in which they are interested. For a popular product, the more obvious examples of this vulnerability may thus be discovered by multiple independent individuals.

Fourth, a new class of 'target' may be found: for example, a rash of vulnerabilities were recently found in the file processing portions of graphics libraries. Between August and October of 2004, twenty-two different vulnerabilities were published in various graphics libraries used by the various POSIX (e.g. Linux) and Microsoft operating systems.[9]

Finally, when a product is newly released, the most obvious vulnerabilities (e.g. those that arise from a reasonably frequent pattern of usage) seem intuitively likely to be discovered by multiple users. Brady *et al* add that when products are initially released a surge of flaws will be identified as the product is used in a more wide range of environments than it was possible for the vendor to test [BAB99]. However, Arora *et al* reason differently; they claim that the socially optimal patching/disclosure times are more lengthy when a product is more new, because attackers have not yet learned the product [ATX04].

Each of these potential causes for rediscovery indicates that the process of vulnerability discovery is heavily biased and that reliability growth modeling tools are thus of limited utility. The truth seems likely to be some degree of deterministic influences on a fundamentally stochastic process.

6 Future Work

In order to fully test the validity of the security growth models described in Section 4, the data set needs to be extended to cover every vulnerability discovered in that operating system. Gathering such data would enable a comparison between versions of OpenBSD as well: for example, is code added after 2.2 more or less secure?

Furthermore, comparison data gleaned from other operating systems would be of even greater benefit. Such data could be used as a metric to compare the security of different programs. Unfortunately, the precision of data collection achieved with OpenBSD depends upon the availability of the source code CVS

[9] CAN-2004-0200, CAN-2004-0597, CAN-2004-0802, CAN-2004-0803, CAN-2004-0817, CAN-2004-0827, CAN-2004-0929 (all buffer overflows), CAN-2004-0599, CAN-2004-0886, CAN-2004-0888, CAN-2004-0889 (all integer overflows), CAN-2004-0691 (heap overflow), CAN-2004-0804 (division by zero), CAN-2004-0598, CAN-2004-0692, CAN-2004-0693 (NULL pointer dereferences), CAN-2004-0687, CAN-2004-0688, CAN-2004-0753, CAN-2004-0782, CAN-2004-0783, CAN-2004-0788 (misc.)

repository. While an equally exacting data collection process is possible with Linux distributions, it will not be possible with closed source systems. Nonetheless, I intend to examine both closed source systems and additional open source systems.

In addition, the incongruity between the assertion by OpenBSD that security patches can often be created and released within hours [Ope05] contrasts rather strongly with the amount of time sometimes taken by Microsoft for its security patches (one security patch in Table 7 was released 758 days after the vendor was initially informed). A significant part of this time differential may be the testing and interoperability requirements that are commercially necessary for Microsoft but that are less pressing for OpenBSD. This difference nonetheless deserves study and could provide useful information for the literature on disclosure policies.

Finally, one way of obtaining information on the number of testers (i.e. the number of individuals looking for vulnerabilities) would be to examine the number of unique posters to the most prominent full disclosure fora: the Bugtraq and Full Disclosure mailing lists. Although these data would not definitively indicate the programs those posters were examining, they could be used to define a relative metric. For example, if twice as many posters were active in 2000 than in 1996, vulnerability interval data could be adjusted accordingly. (Assuming that the ratio of total posters interested in the program remained constant throughout that period.)

7 Conclusion

This work provides evidence for the independent rediscovery of vulnerabilities. It re-examines the software reliability growth models used in prior work and also considers new evidence from deployed software.

The stochastic models used in [Res04] were considered and it is shown that the ICAT data set does not adequately conform to the requirements of these models. The ICAT data set has chronological inconsistencies, uneven coverage, and does not normalize time for effort. A data collection process was created to overcome the former two shortcomings; research is underway on how to compensate for the latter. OpenBSD vulnerabilities were identified through ICAT, Bugtraq, and OpenBSD patch releases. The source code of each vulnerability was then examined, and the CVS repository used to identify both the birth date and the death date of the vulnerability. Thirty months of data (five version releases) were collected. Two models, Yamada's S-Shaped Reliability Growth Model and Brooke's & Motley's Discrete SR Binomial Model provided an adequate fit to the data. They provide estimates that 4.08 and 10.63 faults, respectively, remain in the OpenBSD 2.2 source code.

These models support the hypothesis that the pool of vulnerabilities in OpenBSD 2.2 is being depleted. However, the existence of reliability growth models with an acceptable goodness-of-fit to the data does not conclusively indicate that vulnerability depletion is occurring. Other models, not indicating depletion, may also fit the data. Furthermore, although the data set is clean, it is also small. Nonetheless, these models do indicate the need for further investigation. If vulnerabilities are being depleted, contrary to the results of previous work, then vulnerability hunting may be socially useful: it can help vendors to significantly decrease the number of vulnerabilities in their products.

In addition, more coarse data sources were used to demonstrate in a less mathematically rigorous manner that some non-trivial level of rediscovery is occurring in Microsoft products. Microsoft security bulletins were searched for instances when more than one individual was credited with the discovery of the vulnerability. The independence of the discoveries was then confirmed through private communication with those individuals or their personal security bulletins.

Although the Microsoft security bulletins provide only small quantities of noisy data, they do make a strong case that vulnerabilities have a non-trivial likelihood of being rediscovered. For the three years between 2002–2004, at least 6.58%, 8.51%, and 8.47% of credited vulnerabilities were found to have been independently rediscovered during the relatively short time frame in which Microsoft worked on a patch. If anything, these numbers are conservative and under represent the likelihood of rediscovery. Rescorla's model is thus not an accurate representation of vulnerability finding; on the contrary, full disclosure may indeed be socially optimal. Because the probability of rediscovery is non-zero, identifying vulnerabilities may result in their having been removed before they can be found by attackers, or it may result in vulnerabilities currently exploited by attackers being identified and remediated more quickly. This finding supports Arora *et al*'s model and thus their claim that the optimal behavior for a social planner is to disclose vulnerabilities to the public more rapidly than the vendor would prefer.

Acknowledgements: My thanks go to Eric Rescorla for presenting a new perspective on an interesting problem, for his advice, and for his feedback. I also thank the anonymous reviewers and Ross Anderson for their insight and suggestions. In addition, I gratefully acknowledge the assistance of the individuals who provided me with information about their vulnerability finding efforts: Peter Winter-Smith, Brett Moore, Dustin Schneider, Jouko Pynnon, Jelmer, SPILabs, Renaud Deraison, Cesar Cerrudo, Greg Jones, Ophir Polotsky, Joseph Steinbberg, Noam Rathaus, Michel Trepanier, zenomorph, qFox, Mark Litchfield, David Litchfield, Rodrigo Gutierrez, Roozbeh Afrasiabi, Yorick Koster, and Mike Price.

A Brooke's & Motley's Discrete SR Binomial Model

This model includes the likelihood that faults may be corrected without introducing new ones (α, below). The initial estimate of this probability is one of the parameters determined through maximum likelihood estimation. The probability that there will be n_i errors in the i^{th} test interval is [AIA93]:

$$P(X = n_i) = \binom{\bar{N}_i}{n_i} q_i^{n_i} (1 - q_i)^{\bar{N}_i - n_i} \tag{1}$$

Where $\bar{N}_i$ is the number of faults remaining and subject to discovery at the beginning of test interval i, with the probability α of correcting faults without introducing new faults:

$$\bar{N}_i = \sum\nolimits_{j \epsilon J_i} (w_j N - \alpha N_{i-1,j}) \tag{2}$$

And q_i is the probability of detecting a fault in the i^{th} interval, using K_i test effort:

$$q_i = [1 - (1 - q)^{K_i}] \tag{3}$$

B Yamada's S-Shaped Reliability Growth Model

Yamada's S-Shaped Reliability Growth Model is a finite failure model related to the Goel-Okumoto model [YOO83] [Lyu96]. It considers the fault discovery process to be S-shaped. Initially, few faults are found while the testers learn about the product. This phase is followed by a steep period of exponential growth in discovery as the most easily found faults are identified. Eventually, the reliability of the product increases and the number of faults diminishes.

The probability that n cumulative number of faults will be found by time N^t is [AIA93]:

$$P(N^t = n) = \frac{\mu(t)^n \exp(-\mu(t))}{n!} \quad \text{where} \quad n = 0, 1, ... \tag{4}$$

The mean value function of this model is:

$$\mu(t) = \alpha[1 - (1 + \beta t)e^{-\beta t}] \quad \text{for} \quad \alpha, \beta > 0 \tag{5}$$

The instantaneous rate of change for the expected number of faults with respect to time is known as the failure intensity function. The failure intensity function is the derivative of the mean value function:

$$\lambda(t) = \mu'(t) = \alpha\beta^2 t e^{-\beta t} \tag{6}$$

References

[AGCL86] A A Abdel-Ghaly, P Y Chan, and B Littlewood. Evaluation of competing software reliability predictions. *IEEE Transactions on Software Engineering*, 12(9):950–967, 1986.

[AIA93] AIAA/ANSI. *Recommended Practice: Software Reliability*. ANSI, 1993. R-013-1992.

[AKN+04] Ahish Arora, Ramayya Krishnan, Anand Nadkumar, Rahul Telang, and Yubao Yang. Impact of vulnerability disclosure and patch availability - an emprical analysis. In *Workshop on Economics and Information Security*, May 2004. Minneapolis, MN, USA.

[ATX04] Ahish Arora, Rahul Telang, and Hao Xu. Optimal policy for software vulnerability disclosure. In *Workshop on Economics and Information Security*, May 2004. Minneapolis, MN, USA.

[BAB99] Robert M. Brady, Ross J. Anderson, and Robin C. Ball. Murphy's law, the fitness of evolving species, and the limits of software reliability. Technical Report 471, University of Cambridge Computer Laboratory, September 1999.

[CCR05] Hasan Cavusoglu, Huseyin Cavusoglu, and Srinivasan Raghunathan. How to disclose software vulnerabilities responsibly?, 2005. Under review.

[CER00] CERT/CC. CERT/CC vulnerability disclosure policy, October 2000. http://www.cert.org/kb/vul_disclosure.html.

[CW02] Steve Christey and Chris Wysopal. Responsible vulnerability disclosure process, February 2002. http://www.whitehats.ca/main/about_us/policies/draft-christey-wysopal-vuln-disclosure-00.txt.

[GO79] Amrit L. Goel and K. Okumoto. Time-dependent error-detection rate model for software and other performance measures. *IEEE Transactions on Software Reliability*, R-28(3):206–211, August 1979.

[Goe85] Amrit L. Goel. Software reliability models: Assumptions, limitations, and applicability. *IEEE Transactions on Software Engineering*, SE-11(12):1411–1423, December 1985.

[KT04] Karthik Kannan and Rahul Telang. Economic analysis of market for software vulnerabilities. In *Workshop on Economics and Information Security*, May 2004. Minneapolis, MN, USA.

[Lyu96] Michael R. Lyu, editor. *Handbook of Software Reliability Engineering*. McGraw-Hill, 1996.

[MIT05] MITRE. Common vulnerabilities and exposures, February 2005. http://www.cve.mitre.org/.

[NIS03] NIST. ICAT metabase: A CVE based vulnerability database, September 2003. http://icat.nist.gov.

[Ope98] OpenBSD. CVS – OpenBSD security page, revision 1.12, February 1998. http://www.openbsd.org/cgi-bin/cvsweb/~checkout~/www/security.htm?rev=1.12&content

[Ope05] OpenBSD. OpenBSD security, January 2005. http://www.openbsd.org/security.html.

[PTL93] Joe Palma, Jeff Tian, and Peng Lu. Collecting data for software reliability analysis and modeling. In *CASCON '93: Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering*, volume 1, pages 483–494, 1993. Toronto, Canada.

[Pup01] Rain Forest Puppy. Full disclosure policy (RFPolicy) v2.0, 2001. http://www.wiretrip.net/rfp/policy.html.

[Res03] Eric Rescorla. Security holes... who cares? In *Proceedings of the 13th USENIX Security Symposium*, August 2003.

[Res04] Eric Rescorla. Is finding security holes a good idea? In *Workshop on Economics and Information Security*, May 2004. Minneapolis, Minnesota.

[Sec05] SecurityFocus. Securityfocus HOME vulns archive: Vendor, 2005. http://www.securityfocus.com/bid/.

[Sto03] Walt Stoneburner. SMERFS (Statistical Modeling and Estimation of Reliability Functions for Systems), January 2003. http://www.slingcode.com/smerfs/.

[Wal01] Dolores R. Wallace. Practical software reliability modeling. In *Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop (SEW'01)*. IEEE Computer Society, 2001.

[YOO83] Shigeru Yamada, Mitsuru Ohba, and Shunji Osaki. S-shaped reliability growth modeling for software error detection. *IEEE Transactions on Reliability*, 32(5):475–484, 1983.