Report Draft for Whether Security Problems Should Always Be Publicly Disclosed
(Con)

Introduction to Security Problems and The Potential Effects of Public Disclosure

Some of the crucial factors in assessing software quality are security issues (or defects, or vulnerabilities), their discovery and disclosure, their failure in the field, and their correction. Therefore, whether or not the software developers choose to openly reveal such issues is an excellent approach to assess the software developers, which directly affects the assessment of the program itself. Researchers that study the operational security quality of software systems concentrate on security issue counts and frequencies rather than the operational profile of the program that contains them.[i] But, different behavior when including usage metrics in the security analyses may be seen. On one hand, when observed, the number of security problems disclosed per a set unit time remains approximately constant when viewed from the calendar perspective. This observation could give the illusion that the security quality of the software is most likely not improving. On the other hand, when the number of security problems disclosed per a set unit time is viewed from the usage perspective, an observation might be that the usage of the product is intensifying and yet we are not finding more problems over time. These two points of view prove that an operational view is going to be very helpful and should be considered explicitly when analyzing the security profile of a software product. This observation gives the illusion that not only things are most likely getting better but also there are no more problems being found as time goes on. An operational profile is a description of how frequently a product is used, what it does, and how it operates—including both authorized user usage of the program and attacker activity.[ii] After a security issue is discovered, the latter might become more intense. On the one hand, end-user views of the product's degree of security may increase when disclosure is coupled with a repair or a workaround. On the other hand, end-user views may be less favorable when disclosures are not immediately followed by a solution or a workaround by developers, even when the likelihood that the product would fail due to a security breach may not have altered in a specific scenario. It is essential to take into account these elements in security evaluations when data on software use levels, developer response times, user patching times, and attacker infiltration stages is (publicly) accessible.

Overview of Solution

There were four main objectives when it came to fixing software security issues: create an automation framework to make it easier to gather and compile information on software security issues from public repositories; create and evaluate a state-based process model of the security problem disclosure, disclosure, correction, exploit, failure, and patching; and Create security sub-models as a component of the entire process model to forecast when security concerns will be disclosed, and utilize empirical data and simulations in conjunction with the overall process model to evaluate various disclosure, rectification, and patching procedures.

Data Collection

When it comes to data collection, there are several problems, one of which is the scattered nature of security data and the challenges in gathering that data.[iii] To address this problem, a toolset for the automatic collection of linked data from public repositories was created.[iv] For a wide range of software products, including but not limited to different versions of Firefox, Linux, SeaMonkey, Microsoft Internet Explorer, etc., this toolkit gathered data on both security and non-security issues.

It was discovered that between 0.05 and 5% of the reported issues are security related.[v] For 35% of all security issues, there are open exploits. Only 1.3% of security issues encounter public exploits after being made public, whereas 34% of issues receive exploits even before developers are aware of them and before they are published as a result of an exploit.[vi]

Field failures are caused by between 0.05% and 2% of all software security issues. 95 percent of field failures affect fewer than 50 systems, 3 percent affect between 50 and 1000 systems, and 2 percent affect more than 1000 systems. However, it is known that 50,000–50 million people may have been affected by common security threats during the previous ten years. Only when users engage with the attack mechanism will true software security issues (30% to 80%) fail in practice.[vii] In other words, if consumers are made aware of such problems, they may be prevented.

It has been demonstrated that if a security issue is fully disclosed before a solution is ready, it takes around a week to release the patches needed to fix the issue. The time it takes to issue the fixes needed to fix the failing security vulnerability, in contrast, can take anywhere from a week to a month if full disclosure of the security problem is not made.

Process Modeling

The interaction of events involving users, developers, attackers, security problems, and remedies is described by a model of the reaction to security issues. The model depicts the possible outcomes for a system depending on the identification and disclosure of security flaws, exploits, field failures, different sorts of issues, the level of rectification, etc. By placing a strong emphasis on roles and usage perspectives, the model stands out from other published models and is highly thorough.[viii]

Predictive Modeling

The security problem response approach incorporates predictive modeling. It aids in issue prediction, such as the mean-time-to-problem correcting and the mean-time-to-problem disclosure. One might be able to apply traditional software and system reliability models provided the process remains constant and security concerns behave similarly to non-security problems. However, it is still early in the adoption of such models in the security field.[ix] In order to describe and anticipate some of the transitions in our overall model, two security-oriented models after failing to apply traditional software reliability models on security fault data from a calendar time viewpoint have been investigated. The Bayesian disclosure model and the classical reliability model are these two security-oriented models.

In the context of a system's actual usage, classical reliability models may effectively forecast issue rates per release and across software versions. Software security encompasses both objective and subjective measurements in the Bayesian disclosure model, including implied variability in the operational profile, the impact of cyberattacks, and subjective assessments of exposure and problem severity, among other things. With a relative error of 26%, the Bayesian disclosure model performs at least as well as some other traditional software reliability models in terms of predicting the revelation of security flaws with respect to the operational usage of the program.

Classical Software Reliability Model

Can issue rates be predicted using traditional reliability models in the context of a system's operational use? The Fedora 6 issue exposure rate seems to decrease with the amount of time the system has been in use. Thus, it was discovered that Fedora 6 and other release data may be used to predict the issue exposure rate using Musa's Logarithmic Poisson Execution Time model[x]. In the case of Fedora, many versions show similar traits in terms of reporting and fixing security issues. Based on this, the parameters predicted for a certain software release may also be used for the subsequent release. This might be very helpful when determining whether to switch to a new version and estimating the likelihood that security issues will be disclosed in the upcoming version using the present version.

Bayesian Model

A thorough model of a product's software security attributes must include objective, less objective measurements, subjective assessments of problem exposure and severity, etc. because software security includes both an objective and a subjective component. Two kinds of security disclosures were identified because, as was already established, whether or not disclosures are followed by a patch or workaround will most likely alter how the end user perceives the product's degree of security. The first category includes vulnerabilities that come with solutions, that a vendor distributes once a week, or that the end-user may employ an immediate patch, a problem-avoidance tactic, or a workaround for. The second category is made up of people who need an external solution that will come to a certain amount of time down the road, or potentially never.

It goes without saying that software frequently has flaws when it is originally developed, which are found and fixed as the product gets older. Other errors might, however, be introduced throughout this procedure. Also with security flaws, this is true. Security flaws, on the other hand, suggest criminal intent and exploitation that may purposefully aggravate a relatively harmless software flaw that, by itself, may not result in a software failure. Following the exposure of a flaw, attacks may become more intense, which might alter both the real and perceived likelihood that a computer would have a security breach.

Security problems may be quite subtle, which makes it difficult to discern between them. They may not appear until a correlated set of inputs is present, they may slow down the system, they may provide illegal access to the system and its contents, etc.

The study of the data that was gathered revealed a declining trend in the number of security issues that were reported during the in-service period. It was noteworthy since over the studied period, the systems employed in this study were downloaded millions of times. It was noted that, for the most part, problem reporting and repair systems were keeping up, but that occasionally problems took a very long time to resolve. It is possible for the model to accurately represent this in around 30% of the voluntary situations, assuming immediate fault correction. The median relative error was used to gauge predicted accuracy, much like with classical reliability models. When the predictions are tested after watching the data for around 20% of the execution time, it is often found that a median relative inaccuracy of about 34% has occurred. This demonstrates that the model at least agrees with the collected data. However, when the subjective perspectives that affect the scale and form of the Gamma distribution are taken into account, the actual power of the Bayesian model will begin to emerge.

Numerous research on the software system usage behaviour for various user categories has been conducted in relation to operational use variability. It was also expected that the data from these studies would be suitable for illustrating the operational use of a software's variability. All of this was done in order to do a sensitivity analysis on the prior distributions and prediction observations that were selected.

The estimates of Y and theta, including the mean and the 95% Bayesian confidence interval, differ only a little, according to the findings of the study on the sensitivity of prior distribution selection.

The median relative error, which was 24% in estimating the number of security issues reported, was used to calculate the predictive accuracy for the future.

Based on the findings, it is possible to successfully anticipate security problem disclosure occurrences using the Bayesian security problem disclosure model.

Disclosure and Patching Policies Analysis

Impact of Security Failures

Here, the emphasis was on examining how patching and repair practices in full-disclosure and limited-disclosure browsers affected field failures. It was discovered that "full disclosure" software products require more time to resolve security flaws than "limited disclosure" software products. Therefore, "limited disclosure" solutions would have a bigger overall user impact during the failure of a security issue. However, "full-disclosure" browsers may have more active systems at a given moment than "limited-disclosure" browsers.[xi] What is the total impact when taking into account the usage profile as well as the corrective and patching policies?

In order to provide a response, security breach simulations were created and random constant spread models[xii] designed after the outbreak of the code-red virus were used[xiii]. Four security failure simulation cases were constructed. In the first case, viruses are created to propagate primarily when a substantial fraction of the operating systems are weak. An analysis known as the Mann-Whitney U test[xiv] was utilized to establish whether there is a notable difference in the number of computers impacted by security flaws in "full-disclosure" and "limited disclosure" browsers. On one hand, for full-disclosure browsers, automatic patching was employed. On the other hand, for limited-disclosure browsers, non-automatic patching was used. The findings

showed that the number of systems impacted by security flaws varied significantly. The average ratings demonstrate that more systems are affected by limited disclosure browsers than by full disclosure browsers.

In the second scenario, it is predicted that attackers would continue to target users for exploits even after 80% of users have installed fixes. Therefore, the remaining 20% would be the target of the attackers. This relies on the assumption that security flaws persist until every system has been fixed. It was calculated that all systems with full-disclosure browsers would have received patches after 180 days, while all systems with limited-disclosure browsers would have received patches after 806 days. Although 806 days may seem like a very long period for security flaws to propagate, this may not be the case. For instance, the “Conficker” was a virus that exploited a vulnerability in Microsoft Windows that was patched in October 2008. The “Conficker” virus continued to infect systems for around a year and a half. Getting back to the second scenario, following the required test, it was discovered that there was a considerable variation in the number of systems affected by security flaws. The average ratings demonstrate that more systems are impacted by limited disclosure browsers than by full disclosure browsers.

The users' patching habits in the aforementioned scenarios determine how long a security issue persists in the real world. For "full disclosure" and "limited disclosure" browsers, this would result in varying failure times.

The third case took into consideration the failure window and patching procedure that are common for both full disclosure and limited-disclosure browsers. In this case, it was assumed that the security failures would last for about a month for both full-disclosure and limited-disclosure browsers. The resulting average scores of the impacted systems for limited-disclosure browsers were discovered to be higher than those for full-disclosure browsers.

In the fourth case, the failure periods for popular security threats were observed. Turns out that those failure periods turned out to be approximately a week. This work established the base that there was a need to look at short failure periods. The results of the Mann-Whitney U test show that there is a significant difference in the number of users affected in the event of security failures. The mean scores show that the number of systems affected for “limited-disclosure” browsers is greater than the number of systems affected for “full-disclosure” browsers.

Impact of Patching Policies

Early notification of voluntary security issues may let users take protective measures against prospective cyberattacks. This course of action will raise some questions, such as “what conditions should be present in order to determine that the early disclosure of voluntary security problems is a good policy?”, “How will the developers calculate the number of users that have to take the necessary precautions to compensate for the delay in fixing the presented problem?”, “Is there a way for the developers to take advantage of this opportunity to allow additional time to fix the problem?”, etc. These questions reveal that early disclosure may, have drawbacks. One of those drawbacks is that developers can take advantage of the chance to give themselves more time to address security issues. This can be especially harmful when the developers are unable or fail to calculate the proportion of users who will need to take precautions to make up for the delay in correcting the issues. Following the requisite testing, it was discovered that only 18% or 19% of users take measures, which results in a considerable disparity in the number of people exposed. The sooner the developers can address the security issue, the lower that proportion must remain.

When migrating from a limited disclosure browser to a full disclosure browser, it was discovered that the user would be better off utilizing a limited disclosure browser rather to a full disclosure browser if they are unable than deploy security updates within 5 days after converting to full disclosure.[xv]

Why Security Problems Should Not Always Be Publicly Disclosed

Because most people (over 50%) are technologically illiterate and converting from a limited disclosure browser to a full disclosure browser requires people to apply security patches within 5 days after switching and most people would be too lazy and/or would not know how to apply them, one could conclude that staying with a limited liability browser should be what most people do.

As previously mentioned, publicly disclosing security problems would open the door to more attacks as everyone would know there is a liability in the software – including those who are not aware of such a liability – would be able to exploit that liability to their advantage. As stated, this could easily ruin the operational profile of the software. Not to mention the dangerous and vulnerable position the software users would be put in because of the increased attacks.

Even if the security problem has been patched for most users, attackers would most likely still target the remaining users who did not patch the security problems. Publicly disclosing the security problem increases the probability that those users will be targeted as more attackers would be exploiting the issue in the software.

Conclusion

The security of software is a very important metric when evaluating the quality of the software. If the software system has a large number of users, analyzing the security problems by taking into account the operational profile of the software is prudent when analyzing the operational security of the software systems. Developer response process, user patching process, attacker intrusion processes, all of which are public information, and software usage levels can be collected and analyzed as a single unit. This information can be used to either improve the understanding of security issues, processes, and related decision making or be used to determine better intrusion processes.

Roles and perspectives on software usage should be prioritized in a security problem response model. The classical reliability models perform significantly better when applied to the software security space, where the exposure is stated in terms of in-service time or system utilization, according to the comparison between the classical reliability model and the Bayesian model. In terms of operational software use, the Bayesian Revelation model can forecast the disclosure of security issues with a degree of performance comparable to that of traditional software reliability models.

[i] Bev Littlewood, Sarah Brocklehurst, Norman Fenton, Peter Mellor, David Wright, JohnDobson, John Mcdermid, Dieter Gollmann, and Egham Tw Ex. Towards operational measures of computer security. Journal of Computer Security, 2:211–229, 1993

[ii] John D. Musa, Anthony Iannino, and Kazuhira Okumoto. Software reliability: measurement, prediction, application. McGraw-Hill, Inc., New York, NY, USA, 1987.

[iii] Guido Schryen and Rouven 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.

[iv] Prasanth Anbalagan and Mladen Vouk. On mining data across software repositories. In Proceedings of the Sixth IEEE Working Conference on Mining Software Repositories, May 2009

[v] Prasanth Anbalagan and Mladen Vouk. On mining data across software repositories. In Proceedings of the Sixth IEEE Working Conference on Mining Software Repositories, May 2009

[vi] 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.

[vii] 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.

[viii] Stefan Frei, Martin May, Ulrich Fiedler, and Bernhard Plattner. Large-scale vulnerability analysis. In Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense, LSAD '06, pages 131–138, New York, NY, USA, 2006. ACM.

[ix] Omar H. Alhazmi, Yashwant K. Malaiya, and Indrajit Ray. Measuring, analyzing and predicting security vulnerabilities in software systems. Computers & Security, 26(3):219–228, 2007.

[x] J. D. Musa and K. Okumoto. A logarithmic poisson execution time model for software reliability measurement. In Proceedings of the 7th International Conference on Software Engineering, pages 230–238, Piscataway, NJ, USA, 1984. IEEE Press.

[xi] Stefan Frei, Thomas Duebendorfer, and Bernhard Plattner. Firefox (in) security update dynamics exposed. SIGCOMM Comput. Commun. Rev., 39:16–22, December 2008.

[xii] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to own the internet in your spare time. In Proceedings of the 11th USENIX Security Symposium, pages 149–167, Berkeley, CA, USA, 2002. USENIX Association

[xiii] Cliff Changchun Zou, Weibo Gong, and Don Towsley. Code red worm propagation modeling and analysis. In Proceedings of the 9th ACM conference on Computer and communications security, CCS '02, pages 138–147, New York, NY, USA, 2002. ACM.

[xiv] Brian P. Macfie and Philip M. Nufrio. Applied Statistics for Public Policy. M.E. Sharpe, Inc., New York, USA, 2006

[xv] Anbalagan, Prasanth. A Study of Software Security Problem Disclosure, Correction and Patching Processes, North Carolina State University, Ann Arbor, 2011. ProQuest, https://login.proxy.hil.unb.ca/login?url=https://www.proquest.com/dissertations-theses/study-software-security-problem-disclosure/docview/881634816/se-2.