

The Security of OpenBSD: Milk or Wine?

Andy Ozment & Stuart E. Schechter

About the authors

Andy Ozment is a PhD student in the Computer Security Group at the University of Cambridge. He will graduate in July 2007. This article describes work performed in part while Andy was on the technical staff at MIT Lincoln Laboratory.

Stuart E. Schechter is a researcher in computer security at MIT Lincoln Laboratory. He explores security problems as they relate to system design, economics, and user interfaces. Ironically, Stuart can neither digest milk nor tolerate the taste of wine.

Email addresses

andy.ozment@ieee.org, ses@ll.mit.edu

Disclaimer

This work is sponsored by the I3P under Air Force Contract FA8721-05-0002. Opinions, interpretations, conclusions and recommendations are those of the author(s) and are not necessarily endorsed by the United States Government.

This work was produced under the auspices of the Institute for Information Infrastructure Protection (I3P) research program. The I3P is managed by Dartmouth College, and supported under Award number 2003-TK-TX-0003 from the U.S. Department of Homeland Security, Science and Technology Directorate. Points of view in this document are those of the authors and do not necessarily represent the official position of the U.S. Department of Homeland Security, the Science and Technology Directorate, the I3P, or Dartmouth College.

Introduction

Purchase a fine wine, place it in a cellar, and wait a few years: the aging will have resulted in a delightful beverage, a product far better than the original. Purchase a gallon of milk, place it in a cellar, and wait a few years. You will be sorry. We know how the passing of time affects milk and wine: but how does aging affect the security of software?

Many in the security research community have criticized software developers both for releasing software with so many vulnerabilities and for the lack of any apparent improvement in this software over time. These critics assume that software developers aren't exerting enough effort on writing more secure code. However, we have lacked quantitative evidence that applying effort over time will result in software with fewer vulnerabilities. In short, we don't know whether software is destined to age like milk or has the potential to become wine.

We thus investigated whether or not the rate at which vulnerabilities are reported in OpenBSD is decreasing over time. For a more technical description of this work, see Ozment and Schechter, 2006.

Vulnerability Data

We compiled a database of 140 vulnerabilities reported in the 7.5 years between 19 May 1998 and 17 November 2005. Vulnerabilities were identified by merging data from the OpenBSD security web page and four public vulnerability databases: NVD (formerly ICAT), Bugtraq, OSVDB, and ISS X-Force.

Figure 1 shows the number of vulnerabilities that were ‘introduced’ in each of the fifteen versions of OpenBSD that were released during our study. A vulnerability was introduced in a version if that version is the first to contain the vulnerability within its source code.

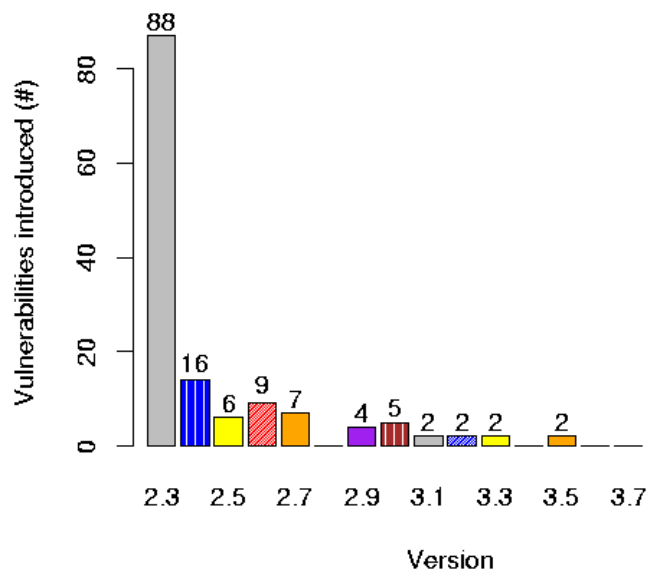


Figure 1: The number of vulnerabilities introduced in each version that was released during the study.

How do we know when a vulnerability was introduced? Vulnerability or patch reports often list the versions affected by that vulnerability; however, vendors and vulnerability hunters rarely bother to test more than two or three versions back. So vulnerability and patch reports are not a sufficiently accurate means of finding the release in which a vulnerability was introduced.

Instead, when vulnerabilities were reported, we used the patch to identify the vulnerable code in the OpenBSD CVS repository. We then tracked the vulnerable code back through all the previous versions of OpenBSD until we could identify when it had first been

checked into the code base. The first release that included that vulnerable code is the release to which the vulnerability is attributed.

Our analysis covers all portions of the OpenBSD code in the OpenBSD team's primary CVS repository. This includes the X-windowing system, the Apache web server, and many additional services not traditionally considered to be part of the core operating system. However, it excludes the 'ports' collection of third-party software, which is not officially part of OpenBSD. We started our study with version 2.3, which we refer to here as the 'foundation version', because it was the first source-code stable release for which all reported vulnerabilities are documented.

During the study, versions of OpenBSD were released approximately every six months. The vulnerabilities that are introduced in each version were usually checked into the CVS repository during the six months prior to that version's release. For example, the vulnerabilities attributed to version 2.4 were introduced in the six months between its release and the release of the prior version. The one exception to this rule is the foundation version (2.3): all vulnerabilities introduced before this version's release are attributed to this version. This includes more than twenty-five years since coding in Berkeley Unix began. As a result, we see in Figure 1 that 62% of the vulnerabilities reported during the study were introduced in the foundation version.

Source Code Evolution

The majority of vulnerabilities reported during the study were thus introduced sometime prior to the release of the foundational version. But now, 7.5 years later, does the security of the foundation version have any relevance to current versions of OpenBSD?

To answer this question, we investigated the proportion of the source code in the most recent version of OpenBSD that remains unchanged since each earlier version. Figure 2 shows the results of our analysis. Each column represents a composite version; each row represents a source version that contributes code to the composite. A line of code in a composite version of OpenBSD is said to originate in a source version if the line was last modified in that source version. The column for version 2.3 is composed of a single row: by definition, all code in this foundation version is said to originate in it. For each successive version, a new row is added to the column to represent the lines of code that were altered/introduced in that release.

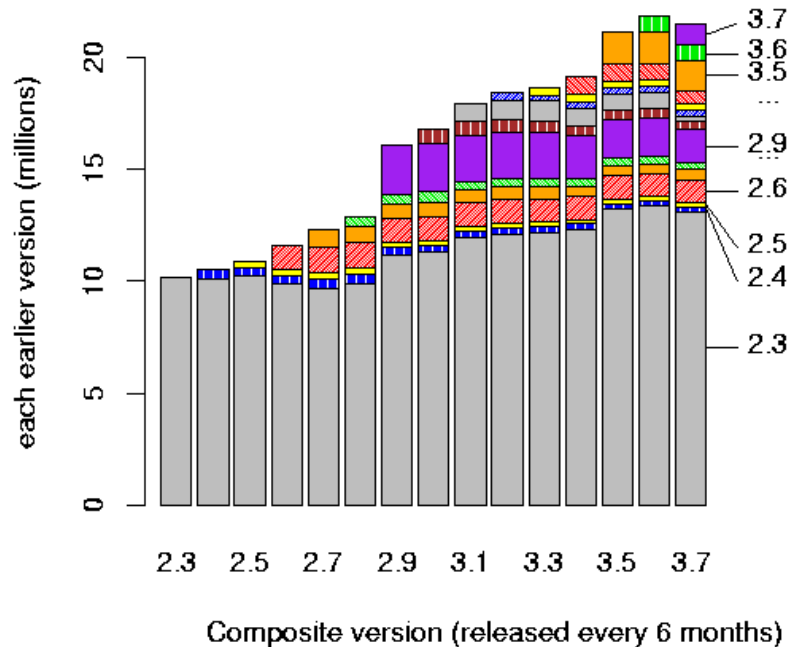


Figure 2: The composition of the full source code. The composition of each version is broken-down into the lines of code originating from that version and from each prior version.

Identifying how the source code evolved over time was a difficult project. We first pre-processed each version of the source code. Only files with the suffix `.c` or `.h` were retained, and all comments were stripped. We then compared each version with each successive version. We used `diff` to compare files with the same path and filename. The `diff` tool was instructed to ignore changes in whitespace or the location of line breaks. Finally, we counted the number of lines in each version that were unchanged from the immediately prior version. By recursively repeating this process, we obtained the data in Figure 2.

The resulting estimate of code commonality is highly conservative. The `diff` tool marked code lines as changed even for trivial alterations like variable renaming and some types of reformatting—and the OpenBSD team has been reformatting the code base. In addition, if a file from a previous version was moved or copied to a new location and if even one line of the file in the new location was changed, our analysis will treat the entire file as new. Furthermore, if the name of a file is changed then all of the code in that file is treated as new. Our comparison results will thus understate the degree to which later releases are composed of substantively unchanged code from earlier releases.

Despite our conservative methodology, Figure 2 shows that unchanged code from the foundation version still comprises 61% of the code in version 3.7—which was released over seven years later. The security of the source code for the foundation version is thus still pertinent to the security of the source code in current versions of OpenBSD.

However, there is another startling result that is visible in Figure 2. The number of lines of source code contributed by the foundation version to each composite version changes over time. That, in itself, is unsurprising. What is surprising is that the number of lines *increases*. How is it that the foundation version contributes more lines of code to version 3.7 than were in the foundation version itself? We discovered that the lines of code derived from the foundation version increases over time because developers reused source code files in different locations. For example, one copy of a compression library file may be used to generate a shared library while another copy of the same file may be used to compile the kernel. Code recycling via source-file replication causes a net increase in the lines of code that are present in later versions.

Several large alterations/introductions of code stand out in Figure 2: versions 2.6, 2.9, and 3.5. The magnitude of the changes in versions 2.6 and 3.5 is primarily due to a large number of files being renamed and slightly altered. Our current methodology thus overstates the number of new lines of code and understates the contribution of code derived from earlier versions. The changes in version 2.9 are caused in part by the renaming of files; however, they were also the result of a major upgrade of the XFree86 package.

Milk or Wine?

Let's return to our original question: does software security improve with age? Unfortunately, we don't have enough vulnerability reports to analyze most versions of OpenBSD. Only the foundation version provides us with enough information: 87 'foundational vulnerabilities' were reported. Our question then becomes: is the rate of vulnerability reporting for OpenBSD version 2.3 decreasing?

We use three different approaches to answering this question. First, we divide the study into two halves and count the number of vulnerabilities reported in each half. Figure 3 shows the result (the confidence intervals assume that vulnerability reporting is a homogenous Poisson process). The number of vulnerabilities reported significantly declines from the first half (58 vulnerabilities) to the second (28 vulnerabilities).

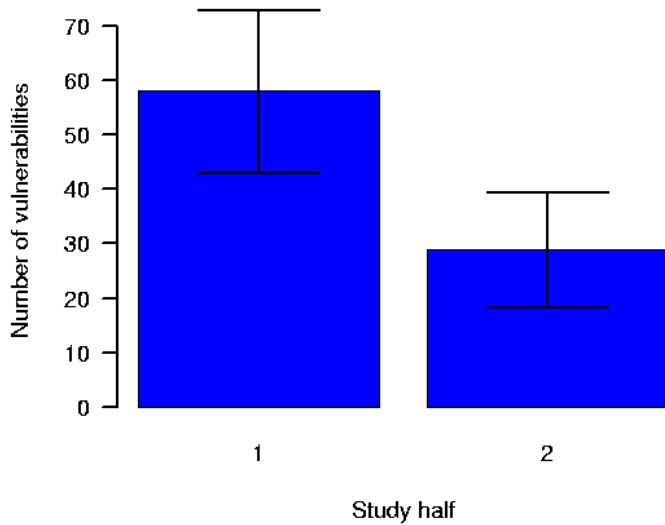


Figure 3: The number of foundational vulnerabilities reported during each half of the study.

The next approach is to utilize a Laplace test, in which the discovery of vulnerabilities is assumed to be a heterogeneous Poisson process. The test assesses whether the inter-arrival times of vulnerability reports are decreasing. We use as our data the number of days elapsed between the identification of each successive foundational vulnerability.

When the calculated Laplace factors are less than lowest horizontal dotted line in Figure 4, the data indicates a decreasing rate of vulnerability reporting with a two-tailed confidence level of 95%. The test finds evidence for a decrease in the rate of vulnerability reporting by the end of year four; by year six, the evidence for a decrease in the reporting rate is statistically significant. This test therefore also supports the conclusion that the rate at which foundational vulnerabilities are reported is declining.

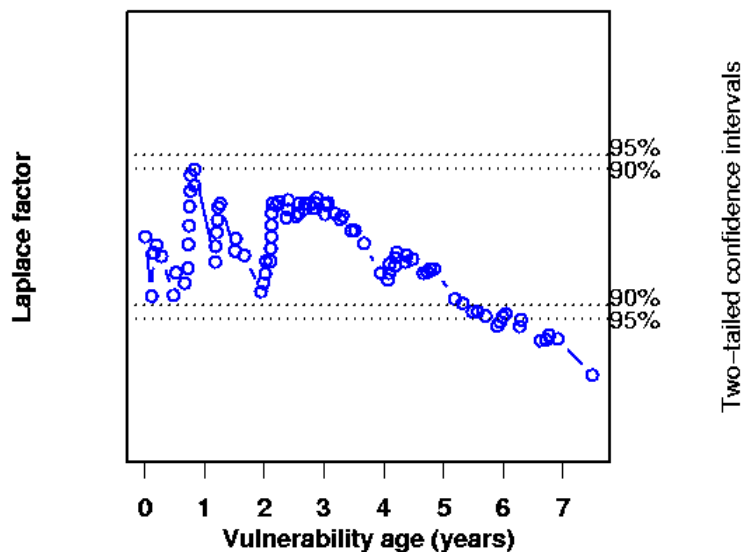


Figure 4: Laplace test for the existence and direction of a trend in the rate of vulnerability reporting.

In our third approach, we attempt to fit reliability growth models to our data. While normally applied to the more random discovery of defects, these models can also be applied to the reporting of vulnerabilities. We analyzed the data with seven time-between-failures reliability growth models. One of the seven models had acceptable one-step-ahead predictive accuracy and goodness-of-fit for the data set: Musa's Logarithmic model. According to this model, the number of vulnerabilities expected to be reported on a given day decreases from 0.051 to 0.024 over the course of the study. Furthermore, it estimates that 67.6% of the vulnerabilities in the OpenBSD 2.3 source code have now been found.

Each of our three approaches thus indicate that the rate of foundational vulnerabilities reported is decreasing.

Caveats

The rate at which vulnerabilities are discovered and reported depends on the level of effort being expended by vulnerability hunters. To measure how much more difficult it has become to find vulnerabilities over time, we would need to normalize the rate of discovery by the effort being exerted and the skills of those exerting it. Unfortunately, vulnerability reports do not include estimates of how many individuals were involved in examining the software, the time they spent, or their relative skills. Our analysis thus can only show that, in the vulnerability hunting environment that existed during the our study, the rate of vulnerability reporting decreased for the foundation version.

While we're at it

Our data prompted us to consider two other questions:

1. What is the median lifetime of a vulnerability?
2. Do larger code changes have more vulnerabilities?

To answer the first question, we calculate the median lifetime of reported vulnerabilities for the foundational version. The median lifetime is the time elapsed between the release of that version and the death of half of the vulnerabilities reported in that version. Alas, we don't know how many vulnerabilities remain in the foundation version or when they will be found. As a result, we can provide only a lower-bound of the median lifetime of vulnerabilities by looking at those vulnerabilities that have been reported. The result is strikingly large: it took 2.6 years to find half of the vulnerabilities in the foundational version that would be found during the 7.5 year study period.

The second question is related to 'vulnerability densities.' Software engineers have examined the defect density of code: the ratio of the number of defects in a program to the number of lines of code. Some have argued that any well-written code can be expected to have a defect density that falls within a certain range, e.g. 3—6 defects per thousand lines of code. Our second question is thus whether or not there is a linear

relationship between the number of lines of code altered/introduced in a version of OpenBSD and number of vulnerabilities introduced in that version.

As we cannot measure the total number of vulnerabilities present, we measure the number discovered within four years of release for each version that is at least four years old. Figure 5 illustrates the relationship between the number of lines of altered/introduced code and the number of vulnerabilities reported. Neither a visual examination of the figure nor Spearman's *rho* test find a correlation between the number of lines of code altered/introduced in a version and the number of vulnerabilities introduced.

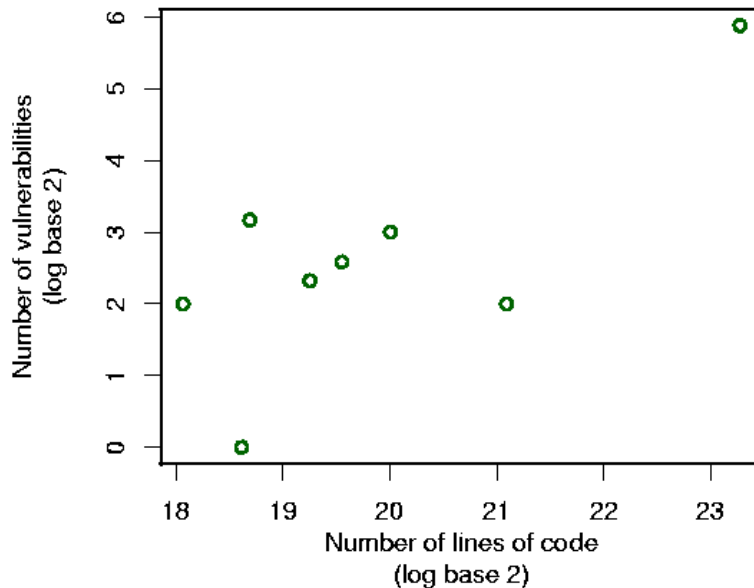


Figure 5: The number of vulnerabilities introduced and reported within four years of release compared to the number of lines of code altered/introduced, by version.

When calculated per thousand lines of code, the density of all *reported* vulnerabilities ranged from 0—0.033 and averaged 0.00657. There appears to be no trend with respect to the densities increasing or decreasing in newer code. These vulnerability densities are thus three orders of magnitude less than normal range of defect densities. However, the two figures are not necessarily contradictory: defects include both vulnerabilities and bugs that are not vulnerabilities. Furthermore, multiple identical security defects that were discovered at the same time are considered a single vulnerability in our data.

Conclusion

Over a period of 7.5 years and fifteen releases, 62% of the 140 vulnerabilities reported in OpenBSD were foundational: present in the code at the beginning of the study. It took more than two and a half years for the first half of these foundational vulnerabilities to be reported.

We found that 61% of the source code in the final version studied is foundational: it remains unaltered from the initial version released 7.5 years earlier. The rate of reporting

of foundational vulnerabilities in OpenBSD is thus likely to continue to greatly influence the overall rate of vulnerability reporting.

We also found statistically significant evidence that the rate of foundational vulnerability reports decreased during the study period. We utilized a reliability growth model to estimate that 67.6% of the vulnerabilities in the foundation version had been found. The model's estimate of the expected number of foundational vulnerabilities reported per day decreased from 0.051 at the start of the study to 0.024. We thus conclude that the foundation version of OpenBSD is wine: it is growing more secure with age.

Reference

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.