Poster: Towards Automated Quantitative Analysis and Forecasting of Vulnerability Discoveries in Debian GNU/Linux

Nikolaos Alexopoulos
Rolf Egert
Tim Grube
Max Mühlhäuser
Technische Universität Darmstadt
{alexopoulos,egert,grube,max}@tk.tu-darmstadt.de

ABSTRACT

Quantitative analysis and forecasting of software vulnerability discoveries is important for patching cost and time estimation, and as input to security metrics and risk assessment methodologies. However, as of now, quantitative studies (a) require considerable manual effort, (b) make use of noisy datasets, and (c) are especially challenging to reproduce.

In this poster abstract we describe our ongoing work towards quantitative analysis of vulnerabilities in Debian GNU/Linux packages. We focus on the challenges of making the process as automated and reproducible as possible, while collecting good-quality data necessary for the analysis. We then state a number of interesting hypotheses that can be investigated, and present preliminary results.

KEYWORDS

software security; vulnerabilities; open-source software; dataset

ACM Reference Format:
Nikolaos Alexopoulos, Rolf Egert, Tim Grube, and Max Mühlhäuser. 2019. Poster: Towards Automated Quantitative Analysis and Forecasting of Vulnerability Discoveries in Debian GNU/Linux. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3319535.3363285

1 INTRODUCTION

As the security community matures and progresses, the need for well-founded security metrics and measurements becomes ever more pressing [1], since the limitations of previous approaches are better understood [5]. Quantitative analysis of the vulnerability discovery process is an important part towards this goal.

Security metrics (e.g., the expected vulnerability discovery rate) serve multiple diverse purposes at the same time: (i) they serve as predictor for patching costs, (ii) they are used as security risk estimator and (iii) they are used to appraise the overall security of products. However, the development, design and verification of such security metrics is a challenging task, which becomes increasingly complicated in complex software systems. Among the most striking challenges that impede the progress of developing security metrics are: (i) the lack of reproducible and reusable quantitative studies, (ii) availability of adequately sized homogeneous datasets and (iii) processes that enable the evaluation and testing of hypotheses to support the design and verification of security metrics.

In this work, we present our ongoing work on quantitative analysis of vulnerabilities in Debian GNU/Linux packages. In particular, we elaborate on our progress towards the automation of the dataset collection process, for generating homogeneous and extensive datasets for vulnerability analysis and forecast operations. In this context, we present a framework that combines a variety of data sources to generate reproducible datasets, and leverage them to enable hypothesis testing.

2 THE DATASET

In this section, we introduce our dataset and discuss the challenges we overcame to establish a semi-automated dataset collection process.

2.1 Dataset collection

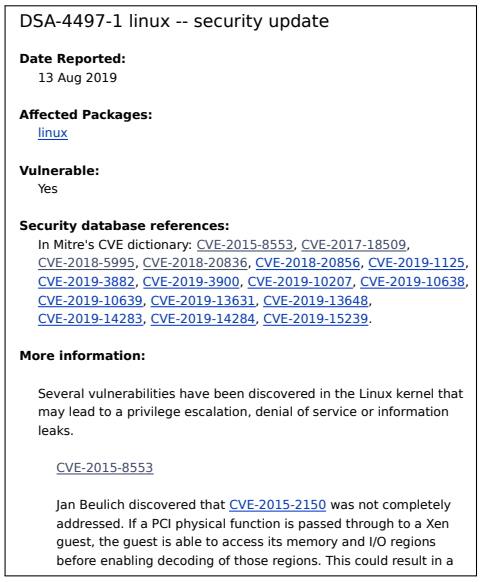
Experience and prior research has shown that poor vulnerability data sources (incomplete, disparate, containing errors) can often lead to wrong conclusions in quantitative studies [2, 6]. Therefore, we need a vulnerability dataset that is "large enough", while uniformly adhering to some common rules and procedures. Critical to the success of our analysis are issues such as which versions of a software component are affected by a given vulnerability. We decided that the data provided by the Debian Security Team[1] are currently the most reliable, large scale, publicly available source of vulnerability information.

2.1.1 Basic vulnerability data. Basic vulnerability information is contained in Debian Security Advisories (DSAs), as shown in Figure 1. The Debian Vulnerability and Analysis Framework[2] provides a way of keeping a local up-to-date database of such security reports and linking them to CVE reports from the National Vulnerability Database (NVD). Thus, related information, such as the type of the vulnerability (according Mitre's CWE), and its severity (CVSS) can be considered.

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3363285

[1] https://www.debian.org/security/
[2] https://github.com/nikalexo/DVAF



DSA-4497-1 linux -- security update

Date Reported:
13 Aug 2019

Affected Packages:
linux

Vulnerable:
Yes

Security database references:
In Mitre's CVE dictionary: CVE-2015-8553, CVE-2017-18509, CVE-2018-5995, CVE-2018-20836, CVE-2018-20856, CVE-2019-1125, CVE-2019-3882, CVE-2019-3900, CVE-2019-10207, CVE-2019-10638, CVE-2019-10639, CVE-2019-13631, CVE-2019-13648, CVE-2019-14283, CVE-2019-14284, CVE-2019-15239.

More information:

Several vulnerabilities have been discovered in the Linux kernel that may lead to a privilege escalation, denial of service or information leaks.

CVE-2015-8553

Jan Beulich discovered that CVE-2015-2150 was not completely addressed. If a PCI physical function is passed through to a Xen guest, the guest is able to access its memory and I/O regions before enabling decoding of those regions. This could result in a

Figure 1: Part of a DSA affecting the Linux kernel.

2.1.2 Source code and churn. Some hypotheses, e.g., investigating how changes between different versions affect the vulnerability discovery rate, require access to specific source code versions. Therefore, an automated way of downloading the source code of the version of the binary that was part of Debian's *testing*[3], was developed. We chose to download the source of the *testing* version because we wanted to investigate when changes were first introduced for testing in the community. We chose to proceed with our analysis in monthly intervals, consequently, a monthly version of the source code would be required per package. To achieve this, we made use of the Debian snapshot project[4], which is a wayback machine that allows accessing chronologically preceding states of Debian, starting from 2005. The developed script repeatedly changed the *sources.list* file accordingly and used the PYTHON-APT library to perform *time-travel* and download the source code of the package as it appeared in Debian repositories on a given date in the past. Considering that some packages had their name changed during their history, special care was taken to consider this modifications by creating a list of synonymous packages. A slightly modified version of PKGDIFF[5] was then used to generate data regarding changes between successive versions, counted in number of bytes added/deleted/modified.

2.1.3 Popularity. To investigate potential hypotheses regarding the relation between the popularity of a software component and its vulnerability discovery rate, we turned to the Debian Popularity Contest[6]. This project collects anonymous statistics on package usage in Debian by volunteers who report their personal usage. Although the current usage data are publicly available, we had to contact the Debian team to get the historical versions of those data. A script was developed to parse the raw data and convert them to a format matching the rest of the available data.

[3]Debian's release cycle consists of the stable, testing and unstable releases. Stable is the only release recommended for commercial use and only vulnerabilities in stable are reported in the DSAs
[4]https://snapshot.debian.org/
[5]https://lvc.github.io/pkgdiff/
[6]https://popcon.debian.org/

2.2 Dataset summary

We decided to populate the dataset with data for a selection of seven popular Debian packages. These were the Linux kernel, two web browsers (Firefox and Chromium), PHP, OpenJDK, Thunderbird, and Wireshark. For each of those packages, there exists one data point per month from March 2005 until December 2018. Each data point contains the following properties: the number of vulnerabilities (also classified as number of low/med/high severity and number per cwe type), popularity stats (installed/used recently/old), churn stats (bytes added/deleted/changed in comparison to previews month).

3 HYPOTHESES

There is a wealth of hypotheses that could be investigated based on our dataset. Some examples follow:

H1 Are vulnerability rates constant over time?
H2 Can we forecast vulnerability discoveries?
H3 Do updates lead to an increased number of discoveries?
H4 What is the effect of package popularity?

In the next section, we move on to present some preliminary results concerning H2 and H3.

4 PRELIMINARY RESULTS

We present preliminary results concerning two hypotheses: (a) can we forecast vulnerability discoveries with linear models based only on prior reports?, and (b) is there statistical causation between update magnitude and discoveries? All of the results presented below are generated via scripts running in python notebooks.

4.1 ARIMA forecasting

To produce forecasts for vulnerability rates, we employ autoregressive integrated moving average (ARIMA) models. These models are very well studied and widespread in econometrics. We fit the models to each time-series individually following the Box-Jenkins approach, and present the results for Linux and OpenJDK in Figure 2. Specifically, we fit the models with data up until July 2018 and test them on unseen data from the last 6 months of 2018. Although we have a phenomenally good fit of the fitted ARIMA model, we should not stop there, as investigating the potential causes of different discovery patterns among packages may lead to interesting observations.

4.2 Code churn as a potential cause

To investigate potential causal relation (not mere correlation) of updates (bytes added) and vulnerability discoveries, we utilize the Granger causality test on the investigated time-series. Results are summarized in the table below:

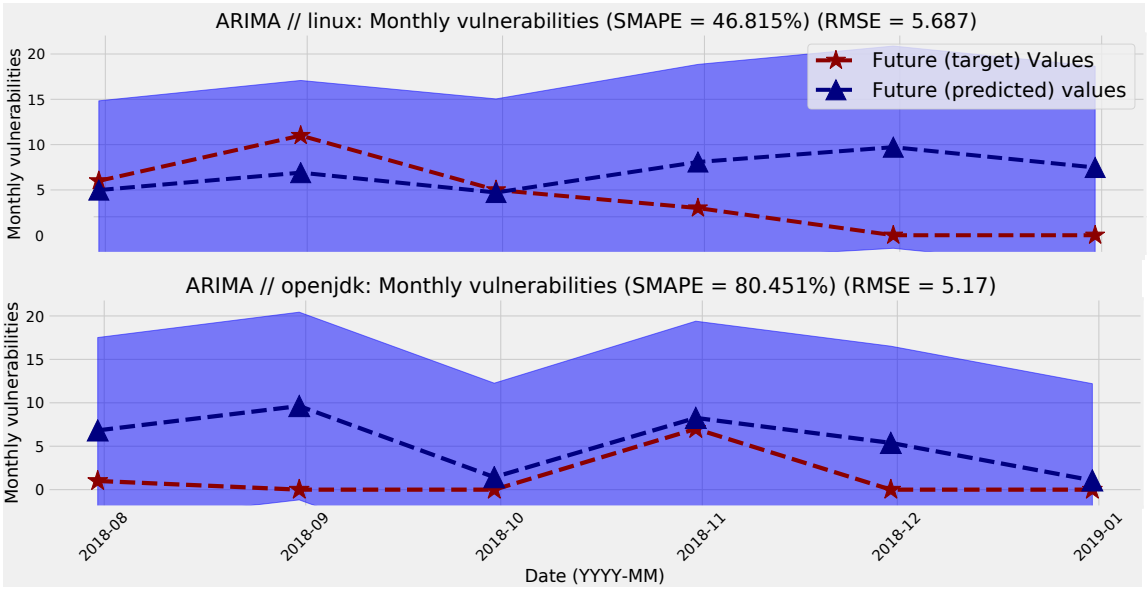


Figure 2: Forecast results of ARIMA models for Linux and Openjdk with 95% confidence intervals.

Package	Test statistic	Critical value	p-value
Linux	2.041	7.815	0.564
Firefox	6.673	7.815	0.083
Chromium	2.477	5.991	0.290
Wireshark	1.898	7.815	0.594
PHP	1.545	11.07	0.908
OpenJDK	12.20	11.07	**0.032**

We observe that only OpenJDK has a statistically significant (with 5% significance level) linear potentially-causal relationship between number of bytes added to subsequent versions and vulnerabilities discovered in the stable release. For the other packages under investigation, there is no such significant relationship between the two variables. For Linux this was expected, as the update cycle of the kernel is continuous rather than discrete. For the other cases, a deeper investigation is necessary.

5 RELATED WORK

Our work is most closely related to the work of Roumani et al. [4] and Pokhrel et al. [3]. In the two aforementioned works, the authors use linear (ARIMA) or non-linear (feed-forward neural networks) forecasting techniques on the time-series of historical vulnerabilities of a small selection of software to forecast future vulnerabilities. Our work is significantly novel compared to theirs, as: (a) we create and use a "cleaner" dataset (Debian DSAs vs NVD), allowing us to draw more reliable conclusions; (b) we develop an open-source forecasting and analysis tool – an automated procedure to validate our results; (c) we collect and take into account metadata (popularity, updates/churn) and explore their relation to the vulnerability discovery rate.

6 CONCLUSIONS AND FUTURE WORK

Reproducible quantitative studies are a vital part of security research and practice. In this poster proposal, we presented a brief overview of our work towards reproducible analysis of vulnerabilities in Debian GNU/Linux. One of the early conclusions of our work is that the vulnerability discovery process is generally complex and vulnerability discovery models proposed in the past are valid only for specific datasets that they were tested against.

We are in the process of testing several hypotheses on the data with the ultimate aim of assessing the relevance of used or proposed security metrics, and/or coming up with new metrics that can be empirically validated. One interesting avenue is testing if there exists non-linear relationships in the data via machine learning models.

ACKNOWLEDGMENTS

This work was supported by the BMBF and the HMWK within CRISP.

REFERENCES

[1] Cormac Herley and Paul C Van Oorschot. 2017. Sok: Science, security and the elusive goal of security as a scientific pursuit. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 99–120.

[2] Viet Hung Nguyen and Fabio Massacci. 2013. The (un) reliability of nvd vulnerable versions data: An empirical experiment on google chrome vulnerabilities. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. ACM, 493–498.

[3] Nawa Raj Pokhrel, Hansapani Rodrigo, and Chris P Tsokos. 2017. Cybersecurity: Time Series Predictive Modeling of Vulnerabilities of Desktop Operating System Using Linear and Non-Linear Approach. *Journal of Information Security* 8, 04 (2017), 362.

[4] Yaman Roumani, Joseph K Nwankpa, and Yazan F Roumani. 2015. Time series modeling of vulnerabilities. *Computers & Security* 51 (2015), 32–40.

[5] Vilhelm Verendel. 2009. Quantified security is a weak hypothesis: a critical survey of results and assumptions. In *Proceedings of the 2009 workshop on New security paradigms workshop*. ACM, 37–50.

[6] Su Zhang, Doina Caragea, and Xinming Ou. 2011. An empirical study on using the national vulnerability database to predict software vulnerabilities. In *International Conference on Database and Expert Systems Applications*. Springer, 217–231.