Vulnerability Analysis as Trustworthiness Evidence in Security Benchmarking: A Case Study on Xen.

Charles F. Gonçalves[*][†], Nuno Antunes[*]
[*]University of Coimbra, CISUC, DEI, Portugal - charles@dei.uc.pt, nmsa@dei.uc.pt
[†]Information Governance Secretariat, CEFET-MG, Brazil - charles@cefetmg.br

***Abstract*—Hypervisors govern the resources of virtualized systems and are a crucial component of many cloud solutions. As a critical component, cloud providers should assess the hypervisor's security to mitigate risk before adoption. Ideally, a benchmark should be applied to compare the security of different systems objectively, but security benchmarking is still an open problem. Notwithstanding, the evaluation of the system's trustworthiness has been adopted as a promising approach as part of this complex evaluation process. In this work, we present a vulnerability data analysis of the Xen hypervisor. Additionally, we address the problem of how to apply this analysis results as trustworthiness evidence that can be applied in security benchmarks. Our results present an insightful characterization of Xen's vulnerabilities evaluating their lifespan, distribution, and modeling. We also show that vulnerability data analysis can qualitatively characterize the Xen hypervisor's trustworthiness and possibly reflect the security development efforts into its codebase.**

***Index Terms*—security benchmark, vulnerabilities, hypervisor, virtualization**

I. Introduction

Cloud Computing acceptance is nothing new. Still, the Coronavirus pandemic has accelerated its adoption even between those reluctant companies [1]. This popularity increase usually comes with security backlashes [2] and reinforces the criticality of having means of ensuring and measuring the security of such systems. A critical component of cloud computing is the *Hypervisor* [3], [4]. It is responsible for managing the physical resources among the virtualized machines, and security exploitation in its domain can seriously jeopardize the operation of cloud providers.

Xen [3] rapidly became adopted by many companies [5], and it was the virtualization manager used by Amazon to build its Elastic Cloud beta version [6]. Many works have been done assessing the securities aspects of the Xen hypervisor. The evaluation of Xen's CVE data is presented in [4]. The authors characterize the security threats and attacks of Xen's core components. The evaluation of the robustness of Xen's hypercall is conducted in [7].

Since the hypervisor is a critical component, its adoption should be guided by an objective evaluation of how it meets the target requirements, especially the non-functional as security. Ideally, a security benchmark [8] should be used to contrast the different security attributes of such systems. However, quantify the security of a system is hard [9] and few quantitative metrics have been proposed in the literature. One of them is a metric that accounts the number of vulnerability by system size, known as vulnerability density [10], [11]. This difficult in quantifying the security is one of the reasons that a security benchmark method remains an open problem despite the efforts towards a definitive approach [8], [12], [13].

One aspect that seems to consolidate among the different security benchmark efforts is to split the evaluation process in two-phases [8], [12], [13] (i) *qualification* - to disqualify incompatible systems that do not match the security requirements, and (ii) *trustworthiness assessment* - to actually compare the system securities. One simple qualification criterion is to avoid vulnerable systems.

A typical method of assessing the security of a system is indirectly through the study of its vulnerabilities. Developer, vendors, and maintainers invest time and resources trying to avoid security faults. According with the Xen's Blog [14], during the development of Xen 4.6, the focus was on improving the code quality and security hardening. Such resource spent in increasing the system quality and security can leverage the trust that system's user has on them, and this leads to our first research question: **Can the security development effort spent during a specific version be reflected in the vulnerability discovery process**?

Many efforts were used to characterize and classify the different types and lifecycles of the vulnerabilities [11], [15], how they are explored [16], and whether their historical and contextual data can be applied to predict [17] or model [10] future vulnerabilities. The study of security vulnerabilities are commonly explored using [17]: (*i*) regression techniques for prediction, (*ii*) machine learning techniques, (*iii*) statistical analyses with the help of reliability growth models [11] and vulnerability discovery models (VDM) [10], and (*iv*) time series analysis.

Under the scope of VDM, the authors of [10] propose the *Malaya-Alhazmi Model* (MAM) using the cumulative number of vulnerability discovered through time. According to the authors, the model can capture the market adoption and use of the software, influencing the vulnerability discovery process. The analysis of vulnerabilities birth-death period is explored on [11], [15]. One evaluation is about the *foundational vulnerabilities*, i.e., vulnerabilities inherited from the first version. In [11] 62% of the vulnerabilities were foundational against 31% in [15]. This information shows that a significant fraction of vulnerabilities is due to code that is not part of the frequently updated source. This observation can indicate that the typical pattern of patching and updating the current software is not entirely effective as thought.

If those vulnerabilities analysis could characterize the system's trust, we could use them into a security benchmark, even in a qualitative manner. Our second research question relates to that: **How can vulnerabilities data analysis be used as trustworthiness evidence**?

In this work, we conduct a vulnerability data analysis of the Xen hypervisor addressing the cited research questions. We address the vulnerability lifecycle of Xen (Section III-A), analyze the vulnerability density (Section III-B), and explore the MAM in the hypervisor data (Section III-C).

The remainder of the paper is organized as follows. Section II explains the dataset, how we created it, and its limitations. In Section III we present our contribution on how to use vulnerability data analyses as trustworthiness evidence followed by a discussion (Section IV) of this approach aspects and its limitations (Section IV-A). Finally, Section V concludes the paper.

II. Dataset and Processing Step

The first step in the dataset assembly was to define the Xen versions to analyze. We decided to start our study when Xen stated supported a Linux kernel (version 4.0) as the DOM0: the privileged VM, which runs at the most privileged user level and has full access to the hardware.

We then used the dataset and tools provided by VulinOSS [18] as a starting point from where we would start creating our Xen's dataset. We updated this database with the last version of the National Vulnerability Database (NVD) dataset. Then we needed to understand the security process and support from the Xen Project.

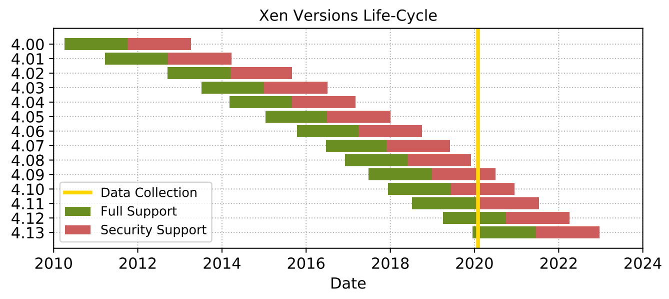


Fig. 1: Xen support lifecycle and its relation with our analysis

According to Xen's documentation, releases are intended to happen roughly every four months and the full support of stable versions last 18 months plus 18 months for security fixes (see Fig. 1). To understand this process is especially essential to comprehend the impact of the development of new releases, where the security fixes happen, which vulnerabilities refer to deprecated versions, among many other aspects.

When dealing with security assessments, we face the challenge of available data to feed our study and its quality. The work in [15] shows that the number of trusted sources and their quality are significant threats for good quality research, even for popular software such as the Mozilla Firefox. Sometimes there is no convergence of approach in reporting vulnerabilities. For instance, the Xen Security Advisory (XSA) 15 was enumerated as the CVE-2012-3496. It was fixed and patched. Later seven new CVEs (CVE-2012-603[0-6]) were created to readdress the CVE-2012-3496 since it was too generic.

To address this data problem we studied the relation between the XSAs and the CVEs to establish a link between them and to not treat the same vulnerability as different. We discovered that not every XSA (24) has a CVE, some XSAs (25) have more than one related CVE and two XSAs relates to the same CVE (XSA-149 and XSA-151 to CVE-2015-7969).

After that, we wanted to link the vulnerabilities data from public repositories with the source code of Xen. We then automated the process of linking CVE/XSA and commits. The developed system parses the Xen XSA page and NVD/CVE database to create a relation between CVE/XSA (as described above). Then it searches commit messages looking for a strong correlation that indicates a fix/patch (e.g., This is CVE/XSA). A second step is to search for a weak correlation showing any relationship with CVE/XSA (e.g., CVE/XSA pattern). The final action is to put all data in a CVE basis and manually validate the output.

To determine the exact lifespan of a vulnerability is almost impossible. At the same time, it is unlikely that any vulnerability is disclosed before a fix [16]. Ozment [11] addresses this lifespan problem analyzing the source code to check when it was fixed (death date) and when it was introduced (birth-date).

The databases as NVD, CVE, and others usually report the "reported date" that cannot always be correlated with the real date of the vulnerability discovery. To address this limitation, we searched related data (References to Advisories, Solutions, and Tools) to find a date reference to the patch that fixes the vulnerability and use it as the approximate discovery date. The estimation process is effective, providing a date more closely related to the time where the vulnerability was "discovered". We found even a difference from 1610 days (CVE-2015-6815 published in 2020).

For the vulnerability death-date, we use the earliest reference between our estimated discovery date and the patch commit date. The birth-date is the earliest affected version's release date and is considered as presented in the release commits, not as described in the documentation.

III. Trustworthiness Evidence

The process of evaluating a system's security should be guided by principles that can help mitigate the risks involved. In this section, we characterize the Xen's vulnerability information using the data described in section II. We then discuss how to use the insights provided by the results as evidence of the system's trust and help in the security characterization of Xen's distinct versions.

A. Vulnerability lifecycle analysis

Many factors can affect the security of a system. In Xen, as in any other hypervisors, we had additional sources of faults as

TABLE I: Xen vulnerability birth-death data. Rows show versions where the vulnerability was fixed and columns versions where the vulnerability was inserted in the codebase. The right part shows the percentage of the overall vulnerabilities that are **L**ocal, **I**nherited from previous versions, and **after-life** vulnerabilities (those that affect obsolete versions)

		The version in which it appear														**Vul. Breakdown** (%)			
		4.00	4.01	4.02	4.03	4.04	4.05	4.06	4.07	4.08	4.09	4.10	4.11	4.12	**Total**	%	I	L	AL
The version in which it was fixed	4.00	1													1	0.4	0.0	0.4	n/d
	4.01	13	4												17	6.7	5.1	1.6	n/d
	4.02	23	6	8											37	14.6	11.4	3.1	n/d
	4.03	9	5	9	1										24	9.4	9.1	0.4	3.5
	4.04	7	5	4	0	15									31	12.2	6.3	5.9	4.7
	4.05	5	2	2	2	1	2								14	5.5	4.7	0.8	2.8
	4.06	7	3	0	5	4	0	6							25	9.8	7.5	2.4	3.9
	4.07	2	0	0	0	2	4	0	7						15	5.9	3.1	2.8	0.8
	4.08	2	0	1	0	0	1	3	0	14					21	8.3	2.8	5.5	1.2
	4.09	0	0	0	0	1	3	1	0	0	18				23	9.1	2.0	7.1	1.6
	4.10	0	0	0	0	0	0	0	1	1	0	9			11	4.3	0.8	3.5	0.0
	4.11	1	1	0	0	0	0	0	0	1	1	0	16		20	7.9	1.6	6.3	0.8
	4.12	0	0	0	0	0	0	0	0	4	0	0	0	11	15	5.9	1.6	4.3	0.0
	Total	70	26	24	8	23	10	10	8	20	19	9	16	11	254	100.0	55.9	44.1	19.3
	%	27.6	10.2	9.4	3.1	9.1	3.9	3.9	3.1	7.9	7.5	3.5	6.3	4.3	100.0				

hardware design problems (CVE-2017-5715 and CVE-2017-5754), hardware wrong specifications (CVE-2019-19581 and CVE-2019-19582) and yet problems that do not have fix either workarounds (CVE-2014-5149). Understanding the lifecycle of known vulnerabilities can shed light on the effort spent on the system by its maintainers and can be interpreted as a measure of the system's trust. For this reason, a careful vulnerabilities lifecycle analysis is advised.

The vulnerability life-span can be measured in the number of versions that it affects. In Fig. 2, we can see the distribution of affected versions of Xen's vulnerabilities. Based on our data, we determine that a vulnerability affects, on average, two versions, but we also have impressive cases that one can affect 11 and 10 distinct versions (Fig. 3).

The vulnerabilities' birth-death period of a system can help understand how the security-related process is distributed in the system lifetime. The TABLE I show many information about this vulnerability lifecycle.

There is no prevalence of *foundational* vulnerabilities on Xen (27.56%). This lack of prevalence might happen by (i) better quality foundational code, or (ii) many changes on codebase during releases which replaced foundational code. Compared with previous works, our result relates better with Firefox [15] (31%) than in BSD [11] (62%).

Adding to the birth-death data in TABLE I, we present the overall percentage of vulnerabilities by version and a further breakdown of different vulnerabilities in the table's right part. A vulnerability is classified as: *local* if it affects only the current version, *inherited* if it was inserted in a previous version, and after-life [15] if it affects versions that are not currently supported, even for security patches. The values showed for local and inherited are complementary; the ones for the after-life are a fraction of the overall percentage.

From a global perspective, the distribution between inherited and local vulnerabilities are statistically equal (55.9% vs. 44.1%). However, the early version tends to have a more prevalence of inherited vulnerabilities than local. This weak tendency changes after version 4.8.

The local classification may inflate the vulnerability count and should be evaluated carefully. The CVE-2012-3516 illustrates this case. It affects version 4.2, but the vulnerability fix date is before the stable version release, i.e., it was never a

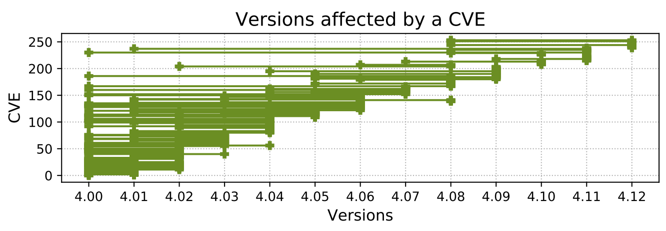


Fig. 2: Vulnerabilities' life-span of studied Xen versions.

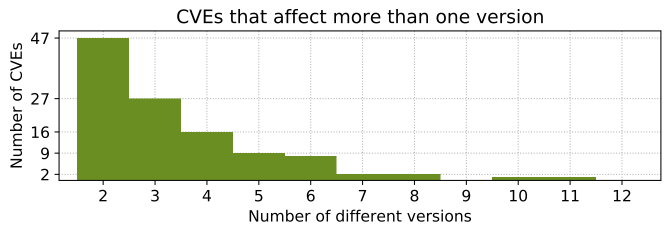


Fig. 3: Vulnerabilities that affect multiple Xen versions.

production threat. Depending on the assessment focus, these cases should be removed from data or classified differently. For instance, it can be interpreted as the software development process's effectiveness to remove security bugs during the system tests. Ideally, the local vulnerabilities should exclude those cases or address them as a different type.

It is interesting to note that 19.3% of the studied vulnerabilities are after-life. This percentage is relevant since Hypervisors are core infrastructural systems and usually do not require new features leading to fewer updates demand.

Finally, we can note in TABLE I that after version 4.6, the absolute number of inherited vulnerability (also the percentual fraction) decreases. This observation can be a sign of the positive effect of the effort spent in security effort mentioned in Xen's Blog.

A benchmark user can use the results presented in this section as a system overall evaluation factor. Using the results above, one can increase or decrease the confidence in the Xen adoption depending on their requirements. Nevertheless, it is clear that these pieces of evidence cannot be considered without additional system evidence. This limitation suggests that these results should be added to a more complex model into a benchmark.

TABLE II: Xen Versions code size, number of Vulnerabilities an its respective *Known* Vulnerability Density (V_{KD})

Ver.	**KLoC**	**#V**	V_{KD}	**Version**	**KLoC**	**#V**	V_{KD}
4.0	445	79	0.1773	4.7	449	29	0.0645
4.1	498	103	0.2065	4.8	455	32	0.0702
4.2	413	119	0.2877	4.9	469	33	0.0704
4.3	396	79	0.1991	4.10	480	19	0.0395
4.4	405	77	0.1897	4.11	488	27	0.0552
4.5	417	60	0.1438	4.12	498	17	0.0341
4.6	418	45	0.1075	4.13	480	1	0.0021

B. Vulnerability Density

In addition to the vulnerabilities lifecycle analysis, we can use the vulnerability density to indicate the success of system maintainers' effort to produce secure code [11]. Vulnerability density (V_D) [10], [11] is the total number of system's vulnerabilities over the system size (see Equation 2). According to [10], V_D can be used to compare systems between versions since they are designed using the same process. Consequently, it is reasonable to expect a small decrease in the V_D as the software evolves. At the same time, is almost impossible to determine the V_D since finding *all* vulnerabilities is not practical. What we can measure is the known vulnerability density (V_{KD}) at a specific moment.

As the system evolve a lower value of V_{KD} is expected for three reasons: *i)* the earliest versions are the ones that have more codebase changes, and statistically, there are more chances of security faults insertions *ii)* the software/project maturity and stability tend to increase, and *iii)* the earliest versions have more testing time (either by a separate process or usage in production).

Again we can see the possible reflection of the 4.6 security and quality effort in TABLE II. During a new version design, developers work on the current version codebase adding functionalities or fixing bugs. Based on this process, we can assume that the security effort during a release development can affect the number of vulnerabilities found in previous (and current) versions. Thus, we can estimate that the effort in version 4.6 had a significant impact on the V_{KD}, approximately 0.8, if we compare the 4.5 to 4.7.

The use of V_{KD} as trustworthiness evidence inside a security benchmark should be qualitative. When evaluating the newest versions, one should expect a similar density as risk estimation. This expectation can guide system administrators to allocate resources to mitigate possible threats. Additionally, we advise that benchmark users evaluate systems and avoid the ones that do not present a decrease in the V_{KD} throughout time, which can roughly translate in an ineffective security quality process during development.

$$y = \frac{B}{BCe^{-ABt} + 1} \quad (1) \qquad V_D = \frac{V}{S} \quad (2)$$

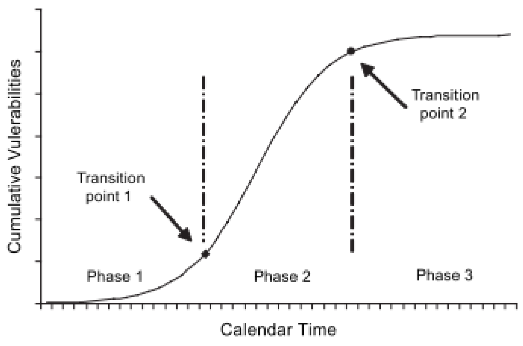
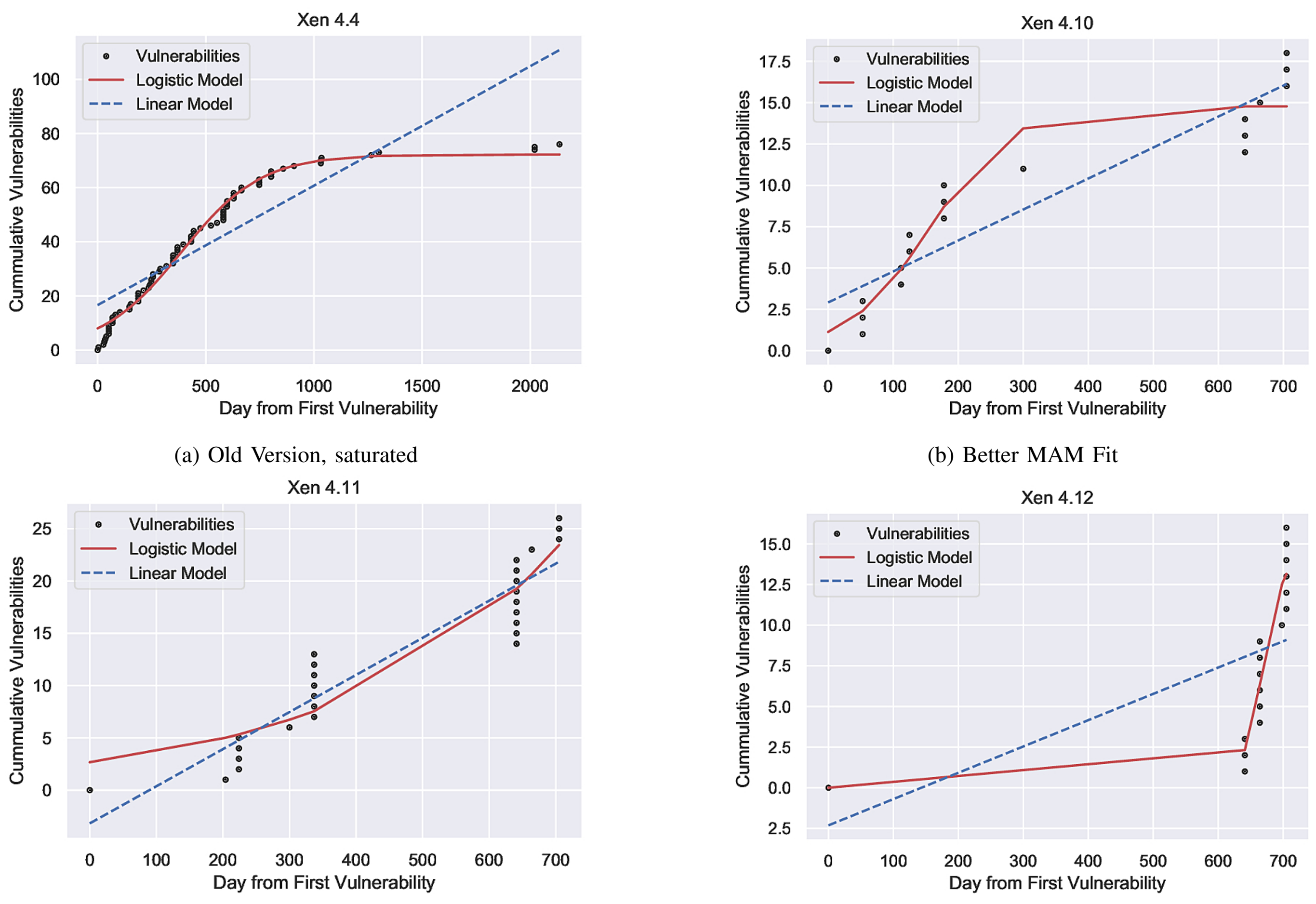


Fig. 4: Three-Phases of the MAM vulnerability discovery process (Image from [10])

C. Saturation Point

The last analysis that we advise is to use *Malaya-Alhazmi Model* (MAM) [10] to determine in which phase any version under evaluation is in this three-phase model (see Fig. 4). The MAM models the cumulative number of vulnerabilities. This time-based logistic model is described by a relation between the time (t), where the vulnerability is discovered, the momentum gained by the market acceptance of the software (A), and the total number of vulnerabilities present in the software (B). The model is given by the Equation 1.

This model tries to capture the market adoption and use of the software (parameter A). Based on [10], a logistic curve describe the vulnerability discovery process that can reflect three different system's phases: (1) early adoption, (2) increase in popularity and acceptance, and finally, (3) popularity limit. According to the authors, in the third phase, the reward for exploring the vulnerabilities starts to decreasing, achieving a



(a) Old Version, saturated

(b) Better MAM Fit

(c) Better Linear Fit

(d) Not saturated

Fig. 5: Current momentum of different versions of Xen based on the MAM. We fitted the data using the non-linear least squares method with a confidence level of 95% (alfa = 5%)

saturation limit. That way, black hat vulnerability finders are more likely to shift effort to a newer version of the software.

Using the vulnerability data of Xen and using the non-linear least squares method (confidence level of 95% / alfa = 5%), we evaluated the model fitting for all versions in our study. We can see in Fig. 5 the distribution of the vulnerability and the fitting curve for the logistic and linear (as a control curve) models from different versions of Xen. For an older version (Fig. 5a), the vulnerability discovery rate is low by its obsolescence, making it less probable that new vulnerabilities would be discovered. At the same time, it comes with all costs related to deprecated software. However, how to test the suitability of newer versions considering the risk associated?

Our recommendations is to avoid the adoption of new versions that has not reached the saturation phase. This can be determined by comparing the distribution of the vulnerability discovery process. If it presents a more linear distribution than the MAM, it should be avoided.

To illustrate this approach, we will evaluate this method in the three most recent versions of Xen. Since version 4.13 was released less than a month (by the data collection time), we can assume that it is in the first phase of the MAM and classify it as untrustworthy evidence. Version 4.12 (Fig. 5d) shows a better MAM fit (P-value 0.991) than the Linear (P-value 0.012)) but can also be assumed a piece of untrustworthy evidence since it is clearly in the second phase of the MAM. Despite the significant fit for both models in the 4.11 version (Fig. 5c), the Linear fit is more likely (P-value 0.990) than the MAM (P-value 0.621) what could also represent untrustworthy evidence. The opposite situation happens for the 4.10 version (Fig. 5b), the MAM model are more likely (P-value 0.998) than the Linear (P-value 0.796), and this version represents trustworthy evidence for presenting a better trade-off when using this criterion.

IV. Discussion

In the previous section we provided some examples of vulnerabilities evaluation methods to use as source of trustworthiness evidence. The natural question that follows is: How to combine those piece of evidence in an practical benchmark evaluation step?

We believe that a security benchmark should follow a two-phase approach as described in [8], [12], [13]. The trustworthiness evidence extracted from the vulnerabilities analysis can be used to feed these two phases. It can match a security requirement for the system qualification or it can be quantitatively inserted into a quality model following a multi-criteria decision-making (MCDM) process [19] to objectively quantify the security of the system under benchmarking.

The type of trustworthiness evidence acquired in this work is qualitative and has limited applicability in the trustworthiness assessment phase. As next step we will work in the quantification of these evidences. To illustrate this possibility we can use the vulnerability density as an example. We can create a model that tries to normalize its value to enable a quantitative comparison between versions. We will need to employ some penalization strategy as function of time to account the adoption time and the latent vulnerabilities.

Even for successful evidence quantification approaches, the use of theirs output as a single source of a system evaluation will probably be misleading. However, they can be combined with additional sources of security/trustworthiness evidence into a quality model and them produce a reliable metric.

A. Threats to Validity

- **Data processing errors**: The dataset build step can add errors in many levels since their creation involves human subjectivity. Even for the automated effort to match all CVEs with all commits, e.g. the commiter can add errors or typos into the commits messages.
- **Subjectivity**: There are a semantic gap between the data and what it represents. Especially in the analyses involved in this work since they cannot be directly contrasted with other systems.
- **Technology Bias**: We are taking some assumptions based only on one system (Xen).
- **Validation**: Some analysis carried here needs additional validation.

V. Conclusion

In this work we employed the results of vulnerability data analysis as a system trustworthiness evidence. We believe that this method can be used in security benchmarks. We also demonstrated how the security development efforts can leave signs on the vulnerability discovery process.

As future work, we have many limitations to address and new analyses to perform. We will explore the information about code changes and features insertion to into a new source of trustworthiness evidence. We will evaluate data from different systems to contrast and validate our results. And finnaly we intend to address the problem to convert those qualitative analyses in a more objective assessment by devising quantitative methods to address the subjectivity of the present method.

Acknowledgment

This work is partially supported by CEFET-MG/Brazil, by the FCT grant `SFRH/BD/144839/2019`, and by the project AIDA (`POCI-01-0247-FEDER-045907`) funded by ERDF – European Regional Development Fund through the Operational Program for Competitiveness and Internationalisation - COMPETE 2020 and by the Portuguese FCT under CMU Portugal. It was developed within CISUC - UID/CEC/00326/2020 funded by the European Social Fund, through the Regional Operational Program Centro 2020.

References

[1] C. Donnelly, "Coronavirus: Enterprise cloud adoption accelerates in face of Covid-19, says research," 2020. [Online]. Available: https://www.computerweekly.com/news/252484865/Coronavirus-Enterprise-cloud-adoption-accelerates-in-face-of-Covid-19-says-research

[2] T. Warren, "Zoom faces a privacy and security backlash as it surges in popularity - The Verge," 2020. [Online]. Available: https://www.theverge.com/2020/4/1/21202584/zoom-security-privacy-issues-video-conferencing-software-coronavirus-demand-response

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *ACM SIGOPS operating systems review*, vol. 37, 2003, pp. 164–177.

[4] R. Patil and C. Modi, "An exhaustive survey on security concerns and solutions at different components of virtualization," *ACM Computing Surveys*, vol. 52, no. 1, 2019.

[5] "Happy XYTH Birthday," Oct 2018. [Online]. Available: https://15anniversary.xenproject.org/

[6] "Amazon EC2 Beta," Aug 2006. [Online]. Available: https://aws.amazon.com/es/blogs/aws/amazon_ec2_beta/

[7] C. F. Gonçalves, N. Antunes, and M. Vieira, "Evaluating the Applicability of Robustness Testing in Virtualized Environments," in *Proceedings - 8th Latin-American Symposium on Dependable Computing, LADC 2018*. Foz do Iguaçu: IEEE, oct 2019, pp. 161–166.

[8] A. A. Neto and M. Vieira, "Selecting secure web applications using trustworthiness benchmarking," *Int. Journal of Dependable and Trustworthy Information Systems (IJDTIS)*, vol. 2, no. 2, pp. 1–16, 2011.

[9] S. M. Bellovin, "On the brittleness of software and the infeasibility of security metrics," *IEEE Security and Privacy*, vol. 4, no. 4, p. 96, 2006.

[10] O. H. Alhazmi, Y. K. Malaiya, and I. Ray, "Measuring, analyzing and predicting security vulnerabilities in software systems," *Computers and Security*, vol. 26, no. 3, pp. 219–228, may 2007.

[11] A. Ozment and S. E. Schechter, "Milk or Wine : Does Software Security Improve with Age ?" *15th USENIX Security Symposium*, pp. 93–104, 2006.

[12] R. A. Oliveira, M. M. Raga, N. Laranjeiro, and M. Vieira, "An approach for benchmarking the security of web service frameworks," *Future Generation Computer Systems*, vol. 110, pp. 833–848, September 2020.

[13] C. F. Gonçalves, "Benchmarking the Security of Virtualization Infrastructures: Motivation and Approach," in *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. Toulouse: IEEE, 2017, pp. 100–103.

[14] "Best Quality and Quantity of Contributions In The New Xen Project 4.6 Release," Oct 2015. [Online]. Available: https://xenproject.org/2015/10/13/xen-4-6/

[15] F. Massacci, S. Neuhaus, and V. H. Nguyen, "After-life vulnerabilities: a study on firefox evolution, its vulnerabilities, and fixes," in *Third International Symposium, ESSoS*, 2011, pp. 195–208.

[16] E. Rescorla, "Is finding security holes a good idea?" *IEEE Security and Privacy*, vol. 3, no. 1, pp. 14–19, jan 2005.

[17] E. Yasasin, J. Prester, G. Wagner, and G. Schryen, "Forecasting IT security vulnerabilities An empirical analysis," *Computers and Security*, vol. 88, p. 101610, jan 2020.

[18] A. Gkortzis, D. Mitropoulos, and D. Spinellis, "VulinOSS: A dataset of security vulnerabilities in open-source systems," *Proceedings - International Conference on Software Engineering*, pp. 18–21, 2018.

[19] M. Martinez, D. De Andres, and J.-C. Ruiz, "Gaining confidence on dependability benchmarks' conclusions through back-to-back testing," in *2014 Tenth European Dependable Computing Conference (EDCC)*. IEEE, 2014, pp. 130–137.