An Empirical Methodology to Evaluate Vulnerability Discovery Models

Fabio Massacci and Viet Hung Nguyen

Abstract—Vulnerability Discovery Models (VDMs) operate on known vulnerability data to estimate the total number of vulnerabilities that will be reported after a software is released. VDMs have been proposed by industry and academia, but there has been no systematic independent evaluation by researchers who are not model proponents. Moreover, the traditional evaluation methodology has some issues that biased previous studies in the field. In this work we propose an empirical methodology that systematically evaluates the performance of VDMs along two dimensions (quality and predictability) and addresses all identified issues of the traditional methodology. We conduct an experiment to evaluate most existing VDMs on popular web browsers' vulnerability data. Our comparison shows that the results obtained by the proposed methodology are more informative than those by the traditional methodology. Among evaluated VDMs, the simplest linear model is the most appropriate choice in terms of both quality and predictability for the first 6 – 12 months since a release date. Otherwise, logistics-based models are better choices.

Index Terms—Software Security, Empirical Evaluation, Vulnerability Discovery Model, Vulnerability Analysis

1 Introduction

Time-based vulnerability discovery models (VDMs) are parametric functions counting the total number of vulnerabilities of a software at an arbitrary time t. For example, if $\Omega(t)$ is the cumulative number of vulnerabilities at time t, the function of the linear model (LN) is $\Omega(t) = At + B$ where A, B are parameters of LN, which are estimated from historical vulnerability data.

VDMs can be seen as a specialization of Software Reliability Growth Models (SRGMs) which were proposed to forecast the total number of defects in a software that could be found as a consequence of testing and debugging procedures [17], [51], [26]. The purpose of a VDM is not to identify vulnerable components (as defect prediction papers do *e.g.,* [29], [42], [15]) but to evaluate the security profile of the software as a whole. Accurate VDMs can be used by software vendors and users to understand security trends, plan patches and updates, forecast security investments, or decide which open source software to bundle with one's own products.

VDMs were initially based on the same principles behind SRGMs. Yet, empirical evidence (in the reported literature and this paper) shows that functions which describe laws for vulnerabilities differ from laws for software bugs SRGMs. For example, Anderson's Thermodynamic (AT) model [8] was one of the first VDM and is explicitly based on SRGM concepts. Yet, it is the worst model in terms of fitting empirical data. Some authors argued that such difference is due to sociological factors [2], [28]: the drop in rate of discovery can be explained by a drop in interest about a software version rather than by the increased difficulty of finding vulnerabilities. Independent experiments showed that many vulnerabilities reported for new versions of a software were indeed present in its previous versions [36], [22].

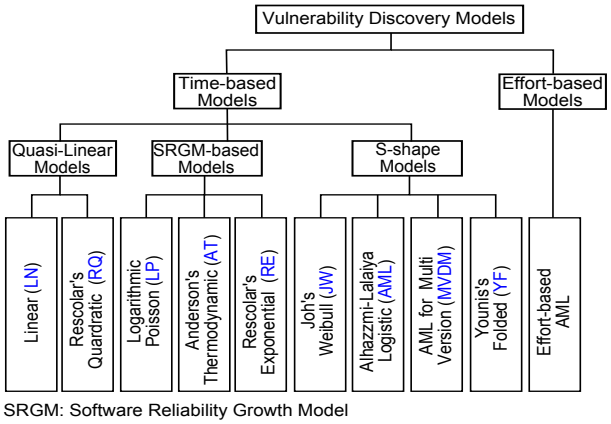


Fig. 1. Taxonomy of Vulnerability Discovery Models.

Fig. 1 sketches a taxonomy of major VDMs, which could be classified into two categories: time-based and effort-based. The former measures the total number of vulnerabilities in the course of time, as previously discussed. The latter counts vulnerabilities regard to the testing effort. This work focuses on time-based VDMs, which were also the major concern of most VDM papers in the literature. Apart from the simplest linear model (LN) [4], and Logarithmic Poisson (LP) model, time-based VDMs (at the time of writing this paper) include Anderson's Thermodynamic (AT) model [8], Rescorla's Quadratic (RQ) and Rescorla's Exponential (RE) models [39], Alhazmi & Malaiya's Logistic (AML) model [2], AML for Multi-version (MVDM) model [19], Weibull model (JW) [18], and Folded model (YF) [53]. Hereafter, we shortly refer to time-based VDM as VDM.

There are two fundamental questions in past VDM papers: *"how well does a VDM fit the data?"*, and *"which model is better than another?"*.

Many studies tried to address these questions but the used methodologies suffer from a number of issues that we document at length in Section 3. In summary:

1) Most past studies do not clearly specify what counts as *one* vulnerability. While the conceptual notion of vulnerability as a software bug is well understood [20], [35], VDM papers counted entries in vulnerability databases. Example 1 and Fig. 2(a) in Section 3.2 show how the same conceptual vulnerability could count as 1 or 3 depending on which database is chosen. Different counts may favor different VDMs.
2) Some studies (*e.g.,* [47], [49]) considered all versions of a software as a single entity. They belong to a product line, but differ by non-negligible amount of code. Considering them as a single entity makes the evaluation imprecise (*e.g.,* Fig. 2(b)).
3) The methodology used in the literature estimates the parameters of a VDM by using all available vulnerability data (at the time of the writing of the paper). This is in sharp contrast with defect prediction techniques in software engineering (*e.g.,* [29], [42], [16]) where only a *part* of the data is used for fitting, another part is used for validation (*e.g.,* cross-validation, next-release validation). For example, while fitting AML to the data set of Win2K vulnerabilities, the experiment in [6] reported a significance level $p\text{-}value = 0.44$, which could be positive; whereas $p\text{-}value$ is 0.05 in [7], which is definitely bad. Barring errors, this can only be explained by the (misleading) methodology.
4) The *p-value* of a statistical test tells us the chance that the estimated model is different from the data. If $p\text{-}value \geq 0.05$, past papers concluded the model well fits the data. Lower *p-value*-values can be soundly used to reject a bad model, but being barely over the threshold is overly optimistic to conclude a model is good. Further, a model is not just "valid" or "invalid". Models have evidence in their favor, and evidence against them. Reporting a single value may obscure temporal properties (*e.g.,* best in the first 6 months since release).
5) Moreover, no study used VDMs as a predictor, for example, to forecast data for the next quarter.

We proposed an empirical methodology to assess the empirical performance of VDMs. The methodology consists of two quantitative metrics, *quality* and *predictability*, and a fully guided process. To evaluate the methodology we apply it to some appropriate domain. In the past, researchers chose operating systems as target applications to evaluate VDMs, but paid little attention to other software classes. Nowadays, web browsers are one of the most important internet applications, and are the products with most vulnerabilities, besides operating systems [52]. According to Google [38] more than two-third of attacks to internet users exploit vulnerabilities of browsers or their plug-ins.

TABLE 1
Performance summary of VDMs.

Model	Performance
AT, RQ	should be rejected due to low quality.
LN	is *the best model* for first 12 months$^{(*)}$.
AML	is *the best model* from $13th$ to $36th$ month $^{(*)}$.
RE, LP	may be adequate for first 12 months $^{(**)}$.
JW, YF	may be adequate from $13th$ to $36th$ month$^{(*)}$.

$^{(*)}$: in terms of quality and predictability for next 3/6/12 months.
$^{(**)}$: in terms of quality and predictability for next 3 months.

We demonstrated the methodology by analyzing the AML, AT, JW, RQ, RE, LP, LN and YF models on Internet Explorer (IE), Firefox, Chrome, and Safari. We selected all major releases which had at least one year of vulnerability data till the collection date.

The experiment reveals interesting findings (TABLE 1): the AT and RQ models are not adequate; the traditional methodology used the maximum horizon of 36 months and allowed a mediocre model (JW/YF) to apparently outperform better but specialized models (LN,AML) that only applied for a limited time span. Such comparison was not done in past studies, because the traditional evaluation methodology did not allow it as VDMs were fitted to a single horizon.

In the next section (§2) we present the terminology. Then we review related work and discusses how they impact our study (§3). Section 4 details our proposed methodology. Section 5 reports the vulnerability acquisition for browser vulnerabilities and Section 6 the evaluation experiment of VDMs on browsers. Section 7 compares the proposed methodology to the traditional one using the experiment results. Last we discuss the threats to the validity (§8) and conclude this work (§9).

2 Terminology

- *A vulnerability* is "an instance of a [human] mistake in the specification, development, or configuration of software such that its execution can [implicitly or explicitly] violate the security policy"[20], later revised by [35]. The definition covers all aspects of vulnerabilities discussed in [9], [10], [13], [40].
- *A data set* is a collection of vulnerability data extracted from one or more data sources.
- *A release* refers to a version of an application *e.g.,* Firefox v1.0. A release is considered as a major release depending on its vendor's version numbering scheme. For example the version numbers of Firefox major releases include 1 decimal digit (*e.g.,* Firefox v1.5, v3.5), whereas the others browsers use only digits before the decimal (*e.g.,* Chrome v1, IE v4). We follow software vendors' decision on what a major release is (see Section 4.1).
- *A horizon* is a specific time interval sample. It is version by the number of months since the release date, *e.g.,* 12 months since the release date.

- *An observed vulnerability sample* (or observed sample, for short) is a time series of monthly cumulative vulnerabilities of a major release since the first month after release to a particular horizon.
- *An evaluated sample* is a tuple of an observed sample, a VDM, and its goodness-of-fit for this sample.

3 Related Work

We recap studies about SRGMs, VDMs, and defect prediction models to highlight similarities and differences. Then we perform a complete review of VDMs studies.

3.1 SRGMs and Defect Prediction Models

SRGM papers try to predict the defect (or bug) discovery rate in a software product by its own developers, as a proxy for the software failure rate [17]. The first and most representative example by Goel and Okumoto [17] has two parameters: the expected total number of defects, and the rate at which the defect discovery rate decreases. The researchers assume that finding bugs become exponentially harder as time goes by. Yamada *et al.* [51] used the gamma function instead of the exponential one. Musa *et al.* [26] proposed the LP model, which assumed an infinite number of bugs. For a discussion about SRGMs, interested readers are referred to [50].

VDMs can be seen as a specialization of SRGMs which focus on security bugs. Yet, this paper and a number of studies [6], [2], [7], [5] showed that VDMs based on SRGMs are empirically inadequate.

The need to consider security defect distinctly from ordinary software bugs has been argued by a number of authors. Roger Needham [28] claimed that security bugs are different from ordinary bugs, "not for a technical but a social reason" because "if a security bug is found in a system there is a community of people who make their personal priority to make the wrong behavior happen, typically in other people's computers". This adversarial process makes the vulnerability discovery process different from normal bug finding as Alhazmi and Malaiya also advocated [2]. At first, people need time to study the software, so they discover few vulnerabilities. When they understand the software, they rapidly discover many vulnerabilities. Finally, the discovery process flattens not because vulnerabilities are harder to find but rather because people lose interest in finding them.

To capture the above phenomenon Alhazmi and Malaiya proposed a logistic, s-shaped model in their seminal paper [2]. The AML model has been evaluated in several applications spanning in various software classes (TABLE 3), such as operating systems [6], [2], [7], [5], server applications [3], [48], [49], and browsers [47]. Our study [22] showed that vulnerabilities found for the current version may, as a by-product, affect earlier versions thus generating an increase of "after-life" vulnerabilities, i.e. vulnerabilities of a version that has long gone out of support. The presence of after-life vulnerabilities may explain why we do not find a strong empirical evidence that the curve may flatten at the end.

The LN model was firstly proposed as a VDM by Alhazmi and Malaiya [4]. In [4], they analyzed AML and LN models in Windows 98/2K and RedHat Linux 7.1. Rescorla [39] proposed RQ and RE models. He evaluated them on WinNT 4.0, Solaris 2.5.1, FreeBSD 4.0 and RedHat 7.0. Rescorla discussed many shortcomings of National Vulnerability Database (NVD), but his study still heavily relied on it. We partially address these shortcomings by taking into account other data sources.

Joh *et al.* [18] proposed JW model, and compared it to AML on WinXP, Win2K3 and Linux (RedHat and RedHat Enterprise). The goodness-of-fit of JW was slightly worse than AML. In other work, Younis *et al.* [53] proposed YF model and compared it to AML on Win7, OSX 5.0, Apache 2.0, and IE8. The paper claimed that YF was somewhat better than AML.

Kim *et al.* [19] introduced AML for Multiple-Version (MVDM), a generalization of AML. It divides vulnerabilities of a version into several fragments. The first fragment includes vulnerabilities affecting this version and past versions, and other fragments include shared vulnerabilities. The authors compared MVDM to AML on Apache and MySQL.

Defect prediction papers aim at predicting whether a source entity (*e.g.*, method, file, component) has a defect. In contrast to SRMGs, they typically try to predict *individual vulnerable components* based on some characteristics of the software components. For instance, some papers used code churn [27], [15], library import patterns [29], code metrics [34], [42], [11], dependencies [54], [32]. These models were often evaluated by two techniques: *cross-validation*, and *next-release (or future) prediction*. The former divides the data into k folds, then uses $k-1$ folds as the training set, and 1 fold as the testing set. The latter, as its name suggested, divides the data by the versions affected by defects, then uses data of older versions as the training set, and data of newer ones as the testing set. Such approach is not used in past VDMs studies.

3.2 An Analysis of the VDMs Evaluation Results

TABLE 2 summarizes the methodology used in VDM evaluation studies whereas TABLE 3 summarizes the evaluation results. In TABLE 2 each study is reported with its target VDMs and the classes of software applications used to conduct the experiment. The table also reports the evaluation methodology of the studies. VDMs have been evaluated in several software classes spanning from server applications, browsers, to operating systems. We briefly discuss some of the issues behind the methodological choices behind these studies.

Vulnerability Count Methods

Most of past studies (see TABLE 2) did not clarify exactly what was *counted as one* vulnerability, and what considered as *one* vulnerable software. Since the purpose

TABLE 2
Summary of VDM evaluation studies.

Akaike Information Criteria (AIC) measures the relative quality of a statistical model for a given data set. Average Error (AE) and Average Bias (AB) measure the average ratios between the actual data and the generated model.

Study	Validated VDM	Software Class	Validation Method		
			Fit Model	GoF test	Predictability
Alhazmi *et al.*[6]	AML	Operating System	Single time horizon	χ^2 test	–
Alhazmi *et al.*[2]	AML, AT, LP, RE, RQ	Operating System	Single time horizon	χ^2 test, AIC	–
Alhazmi *et al.*[7]	AML, LN	Operating System	Single time horizon	χ^2 test	–
Alhazmi *et al.*[3]	AML, LN	Web Servers	Single time horizon	χ^2 test	AE,AB
Alhazmi *et al.*[5]	AML, AT, LN, LP, RE, RQ	Operating System	Single time horizon	χ^2 test, AIC	–
Woo *et al.*[48]	AML	Operating System, Web Servers	Single time horizon	χ^2 test	–
Woo *et al.*[47]	AML	Browser	Single time horizon	χ^2 test	–
Woo *et al.*[49]	AML	Operating System, Web Servers	Single time horizon	χ^2 test	AE,AB
Joh *et al.*[18]	AML, JW	Operating System	Single time horizon	χ^2 test	–
Kim *et al.*[19]	AML, MVDM	DBMS, Web Servers	Single time horizon	χ^2 test	–
Younis *et al.*[53]	AML, YF	Browser, Operating System, Web Servers	Single time horizon	χ^2 test	AE,AB
Rescorla [39]	RE, RQ	Operating System	Single time horizon	unknown	–

of a VDM is to estimate the *number* of bugs, this lack of clarity can make reported results difficult to reproduce.

For example, one vulnerability count could be either an advisory report by software vendors (*e.g.*, Mozilla Foundation Security Advisories – MFSA), or a security bug causing software to be exploited (*e.g.*, Mozilla Bugzilla), or an entry in third-party data sources (*e.g.*, an NVD entry, or CVE, alternatively). A VDM could perfectly fit data counts using a data source, but poorly fit data counts using different data sources.

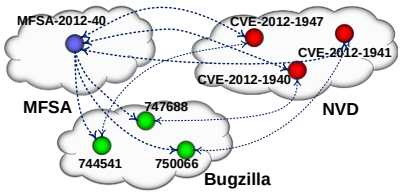
Example 1 In Fig. 2(a), a flaw concerning a buffer overflow in Firefox v13.0 is reported as 1 MFSA entry (MFSA-2012-40), 3 Bugzilla entries (744541, 747688, and 750066), and 3 CVEs entries (CVE-2012-1947, CVE-2012-1940, and CVE-2012-1941). The directional connections illustrate cross references among entries. ■

TABLE 9 later in the paper shows how counting method has a massive impact on the results: by using NVD alone, LN scores 45% of not-fit samples (third-worst), while the YF model makes a 55% of good fits (best of all). By counting number of bugs correlated with NVD entries, *i.e.*, NVD.NBug, the roles are reversed: LN exhibits a 41% of good fits (second best), while YF shows a disastrous 50% of not-fit samples (among the worst).

Some studies [47], [49] assumed all software versions as a single entity, and counted vulnerabilities for it.

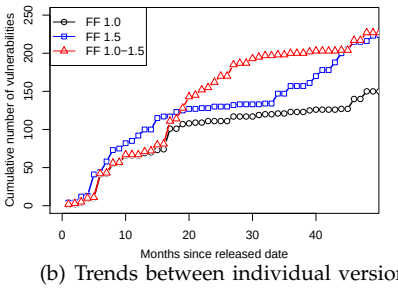
Example 2 Fig. 2(b) visualizes the second fold of this issue in a plot of the cumulative vulnerabilities of Firefox v1.0, Firefox v1.5, and Firefox v1.0-1.5 as a single entity. Clearly, the function of the "global" version is different from the functions of the individual versions. ■

Many VDMs assume that the total number of vulnerabilities in a vulnerable entity (*i.e.*, software) is a time independent constant (*e.g.*, [2], [18], [53]). Considering the combination of all releases as a single entity would



The arrow-headed lines indicates the cross-references between entries. One conceptual flaw is counted differently in three databases. These counts are all legitimate. So what should we choose?

(a) A conceptual security flaw in three data sources



This shows the cumulative vulnerabilities of Firefox v1.0, v1.5, and v1.0-v1.5 as a single entity. The "global" entity exhibits a trend that is not present in the "individual" versions.

(b) Trends between individual versions and their combination

Fig. 2. The two folds of the vulnerability counting issue.

severely violate this assumption since software keeps evolving to introduce new functionality, and thus keeps introducing new vulnerabilities over time. For example, each Firefox version has a code base, which may differ by 30% or more from the immediately preceding one, and 10% – 30% new vulnerabilities were introduced [22]. The same applies to Chrome [31].

Example 3 AML and JW curves fitted the 'bundled' Linux data set (*i.e.*, all versions as a single entity) but the models fitted for the single versions v6.0 and v6.2 were statistically different [18]. ■

One could consider all 114 versions of Chrome 35.0.1916.1,..., 35.0.1916.114 as separate entities instead of the major version Chrome 35.0. The notion of VDM would be meaningless as every version would have a minuscule lifetime. Further, all minor versions are typically

released mainly to fix (security) bugs in a major release, and are unlikely to introduce new vulnerabilities. Thus, we consider only major releases.

Choice of the Statistical Test

Most of studies (see TABLE 2) shared a common evaluation methodology which fitted VDMs to a single time horizon of collected vulnerability data. They used χ^2 test to determine whether a VDM fits actual data. If the test returned *p-value* ≥ 0.05, they claimed the VDM to be a good fit to the data. TABLE 3 summarizes the evaluation results for VDMs. This table reports the *p-values* returned by the goodness-of-fit test between generated curves and actual data *as reported in the cited papers*. In TABLE 3, *p-values* greater than or equal 0.80 (*i.e.*, good fit) are bold, *p-values* between 0.05 and 0.80 (*i.e.*, what we consider an inconclusive fit, but some authors considered a good fit) are reported in italic. The remaining values with *p-values* lower than 0.05 are normally considered a poor fit by statistical tests. The χ^2 test seems to be the most appropriate one among other goodness-of-fit tests, such as the Kolmogorov-Smirnov (K-S) test, and the Anderson-Darling (A-D) test. The K-S test is an exact test, but only applies to continuous distributions and the parameters of the distribution cannot be estimated from the data. Hence, we cannot apply it to perform the goodness-of-fit test for a VDM. The A-D test is a modification of the K-S test that works for some distributions [33, Chap. 1], but some VDMs violate its assumption.

Some studies [2], [5] employed Akaike Information Criteria (AIC) [1], which measures the relative quality of a statistical model for a given data set, to compare VDMs. However, AIC gives no information about the absolute quality. It thus cannot be used to determine the goodness-of-fit of VDM. Moreover, AIC varies with the number of free parameters of a VDM. A model with more free parameters naturally has an advantage in AIC.

Therefore, we do not use AIC, but rely on the χ^2 test in our methodology because it yields comparable results between traditional analysis and ours (when the horizon is the largest data set available).

The second issue is the choice of the value to claim that a VDM was a good fit to a data set. Most studies reported that a model was a good fit when the χ^2 test returned *p-value* ≥ 0.05. Statisticians use a p-value lower 0.05 to reject a model but it does not mean that models with a higher value are good.

We avoid this pitfall by using the statistical practice of selecting good models with an acceptance threshold of 0.80 [24, Chap.8]. A p-value between the acceptance threshold of 0.8 and the rejection value of 0.05 means that evidence is inconclusive. Since a model may obtain a good fit on some data, a bad fit on other data, and an inconclusive fit elsewhere we propose an *inconclusiveness contribution* factor ω as a mean to study the impact of inconclusive VDM in the quality analysis (§4.2).

Model Fitness as a Single Data Point

Previous studies took a snapshot of vulnerability data, and fitted VDMs to this entire snapshot. The single p-value result of this fitting process was used to directly conclude whether a model was "better" than another. At first, this brittle claim only evaluates a model at a single time point (of writing of the paper) and not how this valuation may change over time. This makes repeatability and comparison of experiments difficult. Second, it might allow mediocre models (just barely adequate throughout a product lifetime) to get a better score that specialized models (very good but only for a part of a product's lifetime).

Example 4 In TABLE 3 AML definitely fitted Win98 in [48], [49] (*p-value* $= 1$, essentially a perfect prediction), but inconclusively fitted it in [6] (*p-value* $= 0.74$) and [7] (*p-value* $= 0.21$). The *p-values* that AML fitted Win2K vulnerabilities in [6], [7] were 0.44 and 0.05, respectively. While the former is inconclusive, the latter is essentially reject (*i.e.*, the model is totally unrelated to the data). ■

This is a clear evidence that the goodness-of-fit of a model is changing over time. So, we report the values of VDM goodness-of-fit as a function of time.

Maximal Horizon vs Training and Test Time Horizons

The traditional procedure for VDMs evaluation (all data is training data) has an additional drawback. It is in sharp contrast with the standard procedure used by defect prediction approaches (separating data into training and testing sets) and does not tell us anything about the ability of VDMs to be a good "law of nature" that is able to predict extent the future.

The predictability of VDMs was also discussed in some studies [3], [49], [53] by exploiting two measures, namely Average Error (AE) and Average Bias (AB) [21]. However the application of AE and AB in these studies was inappropriate. The authors used a VDM fitted to the data observed at time t_{max}, and measured its "predictability" at time $t_i < t_{max}$. In other words, this is hardly 'prediction' in the common sense.

We avoid the above pitfall by analyzing the predictability of VDMs in a natural way. Concretely, we fit a VDM to the data observed at time t_0, and use the fitted model to evaluate against data observed at time $t_j > t_0$.

4 Methodology

In this work, we address the following two questions:

RQ1 *How to evaluate the performance of a VDM?*

RQ2 *How to compare between two or more VDMs?*

TABLE 4 summarizes the key steps of our methodology to answer the research questions, while addressing the issues discussed in the related work section.

TABLE 3
Summary of VDM evaluation results (*p-values*) in the literature.

This table reports the returned *p-values* for the goodness-of-fit tests. The values are formatted to indicate the goodness-of-fit of the VDM, particularly: **blue,bold**-good fit; *italic*-inconclusive; red-not fit.

			Browser				DBMS	Operating System															Web Servers					
Model	Study	Year	Firefox	IE	IE 8	Mozilla	MySQL	FreeBSD 4.0	OSX 5	RH Fedora	RH Linux 6.0	RH Linux 6.2	RH Linux 7.0	RH Linux 7.1	RHEL 2.1	RHEL 3.0	Win 2K3	Win2K	Win7	Win95	Win98	WinNT 4.0	WinXP	Apache	Apache 1	Apache 2	IIS	IIS 4	IIS 5
AML	[6]	2005																*0.44*		**1.00**	*0.74*	**0.92**	**0.96**						
	[2]	2005										**1.00**								**1.00**			**0.92**						
	[3]	2006																							**1.00**	**1.00**		**1.00**	**1.00**
	[48]	2006																			**1.00**	**1.00**		**1.00**			**1.00**		
	[47]	2006	*0.41*	0.00		**1.00**																							
	[7]	2007										**1.00**		**0.99**				*0.05*		**1.00**	*0.21*	**0.92**	*0.32*						
	[19]	2007					**0.99**																	**1.00**					
	[5]	2008								*0.43*		**1.00**								**1.00**			*0.15*						
	[18]	2008									**0.81**	**0.85**			**0.83**	*0.66*	**0.96**						**0.94**						
	[49]	2011																			**1.00**	*0.65*		**1.00**			**1.00**		
	[53]	2011			*0.73*				**0.98**										*0.70*							**1.00**			
AT	[2]*	2005										0.00								0.00			0.00						
	[5]*	2008								0.00		0.00								0.00			0.00						
JW	[18]	2008									**0.80**	**0.85**			**0.81**	*0.64*	**0.97**						**0.94**						
LN	[3]*	2006																							0.00	**1.00**		0.00	*0.23*
	[7]*	2007										**0.93**		0.00				**1.00**		0.00	**0.95**	0.00	*0.08*						
	[5]*	2008								0.00		0.00								0.00			0.00						
LP	[2]*	2005										**1.00**								0.00			**0.97**						
	[5]*	2008								0.00		0.00								0.00			0.00						
MVDM	[19]	2007					**1.00**																	**1.00**					
RE	[2]*	2005										**0.93**								0.00			**0.96**						
	[39]	2005						*0.29*					*0.33*																
	[5]*	2008								0.00		0.00								0.00			0.00						
RQ	[2]*	2005										**0.92**								0.00			**1.00**						
	[39]	2005						*0.17*					*0.17*									*0.59*							
	[5]*	2008								0.00		*0.26*								0.00			**0.97**						
YF	[53]	2011			**0.98**				**0.94**										**0.97**							**1.00**			

*: the validation experiment is conducted by people who are not (co-)authors of the corresponding model.

4.1 Step 1: Acquire Vulnerability Data

We classify data sources used to study VDMs for software products. In the sequel *software vendor* is the company or institute that develop the software product(s).

- *Third-party advisory* (TADV): is a vulnerability database maintained by a third-party organization (not the software vendor) *e.g.*, NVD, Open Source Vulnerability database (OSVDB), etc.
- *Vendor advisory* (ADV): is a vulnerability database maintained by the software vendor, *e.g.*, MFSA, Microsoft Security Bulletin. Vulnerability information in this DB has been always evaluated by the vendor.
- *Vendor bug tracker* (BUG): is a bug-tracking database, usually maintained by the vendor.

For our purposes, a vulnerability data entry must have the following minimal features:

- *Identifier* (id): is the identifier of a vulnerability.
- *Disclosure date* (date): refers to the date when a vulnerability is reported to the database[1].
- *Vulnerable Releases* (R): is a list of releases affected by a vulnerability.
- *References* (refs): is a list of links to other sources.

Not every feature is available from all data sources. To obtain missing features, we use id and refs to integrate data sources and extract the desired features from secondary data sources.

Example 5 Vulnerabilities of Firefox are collected from three data sources: NVD[2], MFSA, and Mozilla Bugzilla. Neither MFSA nor Bugzilla provides the *Vulnerable Releases* feature, but NVD does. Each MFSA entry has links to NVD and Bugzilla. We combine these data sources to obtain the missing data. ■

TABLE 5 shows the different data sets used in our study: third-party (*i.e.*, NVD), vendor advisory, and vendor bug tracker data sets. The descriptions of these data sets for a *release* r are as follows:

- NVD(r): a set of CVEs claiming r is vulnerable.
- NVD.Bug(r): a set of CVEs confirmed by at least a vendor bug report, and claiming r is vulnerable.

1. The actual discovery date might be significantly earlier than that date and it is difficult to reliably estimate it [41], [25].

2. Other third party data sources (*e.g.*, OSVDB, Bugtraq, IBM XForce) also report Firefox's vulnerabilities, but most of them refer to NVD by the CVE-ID. Therefore, we consider NVD as a representative of third-party data sources.

TABLE 4
Methodology overview.

Step 1 Acquire the vulnerability data

DESC. Identify the vulnerability data sources, and the way to count vulnerabilities. If possible, different vulnerability sources should be used to select the most robust one (*e.g.*, vendor confirmed ones). Observed samples then can be extracted from collected vulnerability data.

INPUT Vulnerability data sources.

OUTPUT Set of observed samples.

CRITERIA

CR1 *Collection of observed samples*

- Vulnerabilities should be counted for individual releases (possibly by different sources).
- Each observable sample should have at least 6 data points.

Step 2 Fit the VDM to observed samples

DESC. Estimate the parameters of the VDM formula to fit observed samples as much as possible. The χ^2 goodness-of-fit test is employed to assess the goodness-of-fit of the fitted model based on criteria CR2.

INPUT Set of observed samples.

OUTPUT Set of evaluated samples.

CRITERIA

CR2 *The classification of the evaluated samples based on the p-value of a χ^2 test.*

- **Good Fit**: *p-value* $\in [0.80, 1.0]$, a good evidence to accept the model. We have more than 80% chances of generating the observed sample from the fitted model.
- **Not Fit**: *p-value* $\in [0, 0.05)$, a strong evidence to reject the model. It means less than 5% chances that the fitted model would generate the observed sample.
- **Inconclusive Fit**: *p-value* $\in [0.05, 0.80)$, there is not enough evidence to neither reject nor accept the fitted model.

Step 3 Perform goodness-of-fit quality analysis

DESC. Analyze the goodness-of-fit quality of the fitted model by using the temporal quality metric which is the weighted ratio between fitted evaluated samples (both *Good Fit* and *Inconclusive Fit*) and total evaluated samples.

INPUT Set of evaluated samples.

OUTPUT Temporal quality metric.

CRITERIA

CR3 *The rejection of a VDM.*
A VDM is rejected if it has a temporal quality lower than 0.5 even by counting *Inconclusive Fits* samples as positive (with weight 0.5). Different periods of software lifetime could be considered:

- 12 months (young software)
- 36 months (middle-age software)
- 72 months (old software)

Step 4 Perform predictability analysis

DESC. Analyze the predictability of the fitted model by using the predictability metric. Depending on different usage scenarios, we have different observation periods and time spans that the fitted model supposes to be able to predict.

INPUT Set of evaluated samples.

OUTPUT Predictability metric.

CRITERIA

CR4 *The observation period and prediction time spans based on some possible usage scenarios (in browsers).*

Scenario	Observation Period (months)	Prediction Time Span (months)
Short-term planning	6–24	3
Medium-term planning	6–24	6
Long-term planning	6–24	12

Step 5 Compare VDMs

DESC. Compare the quality of the VDM with other VDMs by comparing their temporal quality and predictability metrics.

INPUT Temporal quality and predictability measurements of models in comparison.

OUTPUT Ranks of models.

CRITERIA

CR5 *The comparison between two VDMs*
A VDM vdm_1 is better than a VDM vdm_2 if:

- either the predictability of vdm_1 is significantly greater than that of vdm_2,
- or there is no significant difference between the predictability of vdm_1 and vdm_2, but the temporal quality of vdm_1 is significantly greater than that of vdm_2.

The temporal quality and predictability should have their horizons and prediction time spans in accordance to criteria CR3 and CR4. Furthermore, a controlling procedure for multiple comparisons should be considered.

TABLE 5
Formal definition of data sets.

Data set	Definition
NVD(r)	$\{\mathsf{nvd} \in \mathrm{NVD} \mid \mathsf{r} \in \mathsf{R_{nvd}}\}$
NVD.Bug(r)	$\{\mathsf{nvd} \in \mathrm{NVD} \mid \exists \mathsf{b} \in \mathrm{BUG} : \mathsf{r} \in \mathsf{R_{nvd}} \wedge \mathsf{id_b} \in \mathsf{refs_{nvd}}\}$
NVD.Advice(r)	$\{\mathsf{nvd} \in \mathrm{NVD} \mid \exists \mathsf{a} \in \mathrm{ADV} : \mathsf{r} \in \mathsf{R_{nvd}} \wedge \mathsf{id_a} \in \mathsf{refs_{nvd}}\}$
NVD.NBug(r)	$\{\mathsf{b} \in \mathrm{BUG} \mid \exists \mathsf{nvd} \in \mathrm{NVD} : \mathsf{r} \in \mathsf{R_{nvd}} \wedge \mathsf{id_b} \in \mathsf{refs_{nvd}}\}$
Advice.NBug(r)	$\{\mathsf{b} \in \mathrm{BUG} \mid \exists \mathsf{a} \in \mathrm{ADV}, \exists \mathsf{nvd} \in \mathrm{NVD} : \mathsf{r} \in \mathsf{R_{nvd}} \wedge \mathsf{id_b} \in \mathsf{refs_a} \wedge \mathsf{id_{nvd}} \in \mathsf{refs_a} \wedge \mathrm{cluster_a}(\mathsf{id_b}, \mathsf{id_{nvd}})\}$

Note: $\mathsf{R_{nvd}}, \mathsf{refs_{nvd}}$ denote the vulnerable releases and references of an entry nvd, respectively. $\mathsf{id_a}, \mathsf{id_b}, \mathsf{id_{nvd}}$ denote the identifier of a, b, and nvd. $\mathrm{cluster_a}(\mathsf{id_b}, \mathsf{id_{nvd}})$ is a predicate checking whether $\mathsf{id_b}$ and $\mathsf{id_{nvd}}$ are located next together in the advisory a.

- NVD.Advice(r): a set of CVEs confirmed by at least a vendor advisory, and claiming r is vulnerable. Notice that the advisory report might *not* mention r, but later releases.
- NVD.Nbug(r): a set of vendor bug reports confirmed by a CVE claiming r is vulnerable.
- Advice.NBug(r): a set of bug reports mentioned in a vendor advisory report, which also refers to at least a CVE that claims r is vulnerable.

We do *not* use the NVD alone in our studies. We have shown in [31] that it may contain significant errors to the point of tilting statistical conclusions.

An *observed sample* is a time series of (monthly) cumulative vulnerabilities of a release. It starts from the first month since release to a month specified by the experimenter. This time interval is the *horizon* of the sample. A month is an appropriate granularity for sampling because week and day are too short intervals and are subject to random fluctuation. Additionally, this granularity was the same granularity reported by all studies listed in TABLE 2.

Let R be the set of analyzed releases and DS be the set of data sets, an observed sample (denoted as *os*) is a time series (TS) defined as follows:

$$os = \mathsf{TS}(r, ds, \tau) \tag{1}$$

$r \in R$ is a release in the dataset;
$ds \in DS$ is the data set where samples are extracted;
$\tau \in \mathrm{T}_r = [\tau^r_{min}, \tau^r_{max}]$ is the horizon of the observed sample, in which T_r is the *horizon range of release* r.

In the horizon range of release r, the minimum value of horizon τ^r_{min} of r depends on the starting time of the first observed sample of r. Here we choose $\tau^r_{min} = 6$ for all releases so that all observed samples have enough data points to achieve statistical applicability of the fitting test. The maximum value of horizon τ^r_{max} depends on the data collection period.

Example 6 IE v4.0 was released in September, 1997 [45]. The first month was October, 1997. The first observed sample of IE v4.0 is a time series of 6 numbers of cumulative vulnerabilities for the $1st, 2nd, \ldots, 6th$ months. On $30th$ June 2012, IE v4.0 had been released for 182 months, yielding 177 observed samples. The maximum value of horizon ($\tau^{\mathsf{IEv4.0}}_{max}$) is 182. ■

4.2 Step 2: Fit a VDM to Observed Samples

We estimate the parameters of the VDM formula by a regression method. The fitted curve (or fitted model) is

$$vdm_{\mathsf{TS}(r,ds,\tau)} \tag{2}$$

where vdm is the VDM being fitted; $os = \mathsf{TS}(r, ds, \tau)$ is an observed sample from which the vdm's parameters are estimated. (2) could be shortly written as vdm_{os}.

Example 7 Fitting the AML model to the NVD data set of Firefox v3.0 at the $30th$ month, *i.e.*, the observed sample $os = \mathsf{TS}(\mathsf{FF3.0}, \mathrm{NVD}, 30)$, generates the curve:

$$AML_{\mathsf{TS(FF3.0,NVD},30)} = \frac{183}{183 \cdot 0.078 \cdot e^{-0.001 \cdot 183 \cdot t} + 1}$$

Fig. 3 illustrates the plots of three curves $AML_{\mathsf{TS}(r,\mathrm{NVD},30)}$, where r is FF3.0, FF2.0, and FF1.0. ■

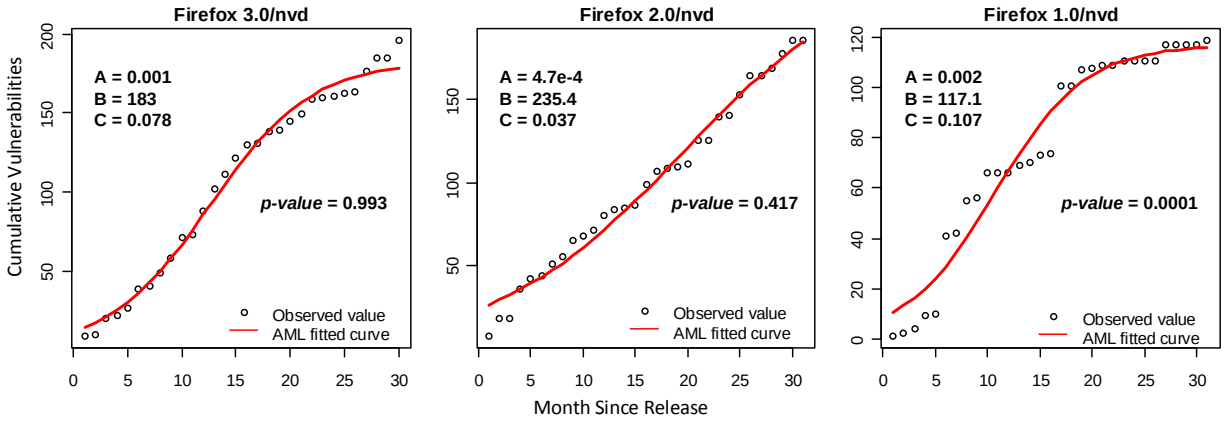
To measure the goodness-of-fit, we employ Pearson's Chi-Square (χ^2) and calculate the χ^2 statistic value of the curve by using the following formula:

$$\chi^2 = \sum_{t=1}^{\tau} \frac{(O_t - E_t)^2}{E_t} \tag{3}$$

where O_t is the observed cumulative number of vulnerabilities at time t (*i.e.*, t^{th} value of the observed sample); E_t denotes the expected cumulative number of vulnerabilities (the value of the curve at time t). The larger χ^2, the smaller goodness-of-fit. If the χ^2 value is large enough, we can safely reject the model. The χ^2 test requires all expected values be at least 5 to ensure the validity of the test [33, Chap. 1].

The conclusion whether a VDM curve statistically fits an observed sample relies on the *p-value* of the test, which is derived from χ^2 value and the degrees of freedom (*i.e.*, the number of months minus one). The *p-value* is the probability that we wrongly reject the *null hypothesis* when it is true (*i.e.*, error Type I). The null hypothesis used in past research papers is that *"the model fits the data."* [7, page 225]. Therefore, if the *p-value* is less than the significance level α of 0.05, we can reject a VDM because there are less than 5% chances that this fitted model would generate the observed sample. This provides us a robust test to *discard* a model.

To consider acceptable a VDM, we use as a threshold the power of the χ^2 test, the probability of rejecting the null hypothesis when it is indeed false (the complementary of the probability of committing a Type II error). Normally, 'an 80% power is considered desirable' [24, Chap. 8]. Hence we *accept* a VDM if the *p-value* is greater than or equal to 0.80, we have more than 80% chances of generating the observed sample from the fitted curve. In all other cases, we neither accept nor reject the model (inconclusive fit). The criteria CR2 in TABLE 4 summarizes the assessment based on the *p-value* of the χ^2 test.



A,B,C are three parameters in the formula of the AML model: $\Omega(t) = \frac{B}{BCe^{-ABt}+1}$ (see also TABLE 8). At the horizon 30 months, AML fits well Firefox v3.0 vulnerabilities, and might fit Firefox v2.0, whereas, it poorly fits Firefox v1.0.

Fig. 3. Fitting the AML model to the NVD data sets for Firefox v3.0, v2.0, and v1.0.

In the sequel, we use the term *evaluated sample* to denote the triplet composed of an observed sample, a fitted model, and the *p-value* of the χ^2 test.

Example 8 In Fig. 3, the first plot shows the AML model with a *Good Fit* ($p\text{-value} = 0.993 > 0.80$), the second plot exhibits the AML model with an *Inconclusive Fit* ($0.05 < p\text{-value} = 0.417 < 0.80$), and the last one denotes the AML model with a *Not Fit* ($p\text{-value} = 0.0001 < 0.05$). To calculate the χ^2 test we refit the model every time. ■

4.3 Step 3: Goodness-of-Fit Quality Analysis

We introduce the *goodness-of-fit quality* (or *quality*, shortly) by measuring the overall number of *Good Fit*s and *Inconclusive Fit*s among different samples.

Let $OS = \{\mathsf{TS}(r, ds, \tau) | r \in R \wedge ds \in DS \wedge \tau \in \mathrm{T}_r\}$ be the set of observed samples, the *overall quality* of a model *vdm* is the weighted ratio of the number of *Good Fit* and *Inconclusive Fit* evaluated samples over the total ones:

$$Q_\omega = \frac{|GES| + \omega \cdot |IES|}{|ES|} \tag{4}$$

$ES = \{\langle os, vdm_{os}, p\rangle \, | os \in OS\}$ is the set of evaluated samples generated by fitting *vdm* to observed samples;
$GES = \{\langle os, vdm_{os}, p\rangle \in ES | p \geq 0.80\}$ is the set of *Good Fit* evaluated samples;
$IES = \{\langle os, vdm_{os}, p\rangle \in ES | 0.05 \leq p < 0.80\}$ is the set of *Inconclusive Fit* evaluated samples;
$\omega \in [0..1]$ is the *inconclusiveness contribution* factor denoting that an *Inconclusive Fit* is ω times less important than a *Good Fit*.

The overall quality metric ranges between 0 and 1. The quality of 0 indicates a completely inappropriate model, whereas the quality of 1 indicates a perfect model. Once again, this metric is an optimistic measure because we are "refitting" the model as more data become available.

The factor ω denotes the contribution of an inconclusive fit to the overall quality. A skeptical analyst would set $\omega = 0$: only *Good Fits* are meaningful. An optimistic analyst could chose $\omega = 1$: an *Inconclusive Fit* is as good as a *Good Fit*. The optimistic choice $\omega = 1$ has been adopted by all model proponents.

The value of ω could also be set based on the average *p-value* ($\bar{p}$) of inconclusive cases.

- If $\bar{p} \approx 0.05$, the VDM most likely does not fit the actual data, $\omega = 0$.
- If $\bar{p} \approx 0.80$, most likely the VDM fits the data, $\omega = 1$.

Therefore, we could approximate ω as follows:

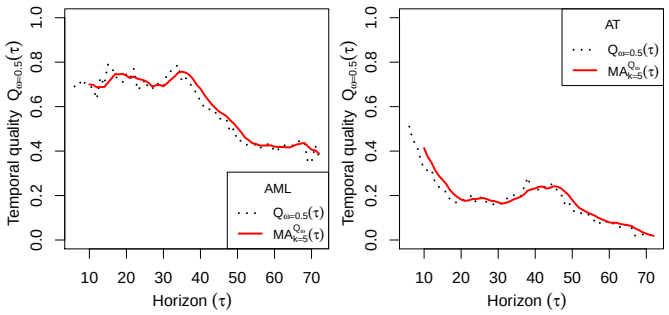
$$\omega \approx \frac{\bar{p} - 0.05}{0.80 - 0.05} \tag{5}$$

where $\bar{p}$ is the average *p-value* of inconclusive evaluated samples. We have analyzed about $6,100$ inconclusively evaluated samples, the average *p-value*: $\bar{p} = 0.422$. Replacing this value in (5), we obtain $\omega \approx 0.5$. It is consistent with the intuition that an *Inconclusive Fit* is as half-good as a *Good Fit*.

Example 9 Among $3,895$ evaluated samples of AML for IE, Firefox, Chrome, and Safari, AML has $1,526$ *Good Fits*, $1,463$ *Inconclusive Fits*. The overall quality of AML with different ω thus is: $Q_{\omega=0} = 0.39, Q_{\omega=0.5} = \frac{1,526+0.5\cdot 1,463}{3,895} = 0.58, Q_{\omega=1} = 0.77$ ■

The overall quality metric does not capture alternating performance in time. A VDM could produce a lot of *Good Fits* evaluated samples for the first 6 months, but almost *Not Fits* at other horizons.

To capture this effect, we introduce the *temporal quality* metric which represents the evolution of the overall quality over time. The temporal quality $Q_\omega(\tau)$ is a function that return the weighted ratio of the *Good Fit* and *Inconclusive Fit* evaluated samples over total sample



Dots represent the data points, whereas the solid line is the moving average. The AML quality is greater than 50% till month $36th$. AT's quality is low.

Fig. 4. Temporal quality of AML and AT models.

for each value of the horizon τ.

$$Q_\omega(\tau) = \frac{|GES(\tau)| + \omega \cdot |IES(\tau)|}{|ES(\tau)|} \tag{6}$$

$\tau \in \mathrm{T}$ is the horizon that we observe samples, in which $\mathrm{T} \subseteq \bigcup_{r \in R} \mathrm{T}_r$ is the subset of the union of the horizon ranges of all releases r in evaluation;
$ES(\tau) = \{\langle os, vdm_{os}, p\rangle \,|os \in OS(\tau)\}$ is the set of evaluated samples at the horizon τ; where $OS(\tau)$ is the set of observed samples at the horizon τ of all releases;
$GES(\tau) \subseteq ES(\tau)$ is the set of *Good Fit* evaluated samples at the horizon τ;
$IES(\tau) \subseteq ES(\tau)$ is the set of *Inconclusive Fit* evaluated samples at the horizon τ;
ω is the same value used for the overall quality Q_ω.

To study the trend of the temporal quality $Q_\omega(\tau)$, we use the *moving average* to smooth out short-term fluctuations. Each point in the moving average is the average of adjacent points in the original series.

$$MA_k^{Q_\omega}(\tau) = \frac{1}{k}\sum_{i=1}^{k} Q_\omega(\tau - i + 1) \tag{7}$$

where k is the *window size*. The value k is less than the minimum horizon ($k \leq \tau^r_{min}$) for the computation to be possible. Additionally, k should be an odd number so that variations in the mean are aligned with variations in the data rather than being shifted in time.

Example 10 Fig. 4 depicts the moving average for the temporal quality of AML and AT models. We choose a window size $k = 5$ because the minimum horizon $\tau^r_{min} = 6$ and $k = 3$ is too small to smooth out spikes. ■

4.4 Step 4: Perform Predictability Analysis

The predictability of a VDM measures the capability of predicting future trends of vulnerabilities, *i.e.*, its practical relevance. The calculation of the predictability of a VDM has two phases, *learning phase* and *prediction phase*. In the learning phase, we fit a VDM to an observed sample at a certain horizon. In the prediction phase, we evaluate the qualities of the fitted model on observed samples in future horizons.

Let $vdm_{\mathsf{TS}(r,ds,\tau)}$ be a fitted model at horizon τ. The prediction quality of this model in the next δ months (after τ) is calculated as follows:

$$Q^*_\omega(\tau, \delta) = \frac{|GES^*(\tau, \delta)| + \omega \cdot |IES^*(\tau, \delta)|}{|ES^*(\tau, \delta)|} \tag{8}$$

$ES^*(\tau, \delta) = \{\langle \mathsf{TS}(r, ds, \tau + \delta), vdm_{\mathsf{TS}(r,ds,\tau)}, p\rangle\}$ is the set of evaluated samples at the horizon $\tau + \delta$ in which we evaluate the quality of the model fitted at horizon τ ($vdm_{\mathsf{TS}(r,ds,\tau)}$) on observed samples at the future horizon $\tau + \delta$. We refer to $ES^*(\tau, \delta)$ as set of evaluated samples of prediction;
$GES^*(\tau, \delta) \subseteq ES^*(\tau, \delta)$ is the set of *Good Fit* evaluated samples of prediction at the horizon $\tau + \delta$;
$IES^*(\tau, \delta) \subseteq ES^*(\tau, \delta)$ is the set of *Inconclusive Fit* evaluated samples of prediction at the horizon $\tau + \delta$.
ω is the same as for the overall quality Q_ω.

Example 11 Fig. 5 shows the prediction qualities of two models AML and AT starting from the horizon of $12th$ month ($\tau = 12$, left) and $24th$ month ($\tau = 24$, right), and predicting the value for next 12 months ($\delta = 0 \ldots 12$). ■

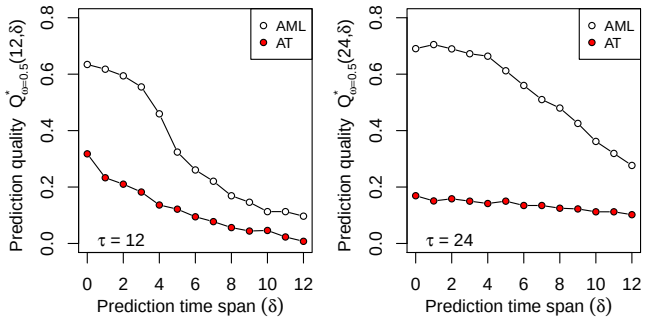
In planning, the idea of 3-6-12-24 month rolling plans has been widely adopted in many fields such as banking, clinical trials, and economic planning. We report the predictability of VDMs in next 3, 6, and 12 months, but not in next 24 months because all VDMs perform badly. Assume a new version is shipped every quarter, we could envisage the following illustrative scenarios:

- *Short-term planning* (3 months): we want to predict the trend in the next quarter to allocate resources for fixing vulnerabilities.
- *Medium-term planning* (6 months): we are looking the next 6 months to decide whether keeping the current system or updating it.
- *Long-term planning* (12 months): we would like to predict vulnerability reports to decide whether to select the software for inclusion in a product with a longer lifetime.

We assess the predictability of a VDM not only along the prediction time span, but also along the horizon to ensure the VDM is able to consistently predict the vulnerability trend in a desired period. For this purpose we introduce the *predictability* metric which is the average of prediction qualities at a given horizon.

The predictability of the curve vdm_{os} at the horizon τ in a time span of Δ months is defined as the average of the prediction quality of vdm_{os} at the horizon τ and its Δ consecutive horizons $\tau + 1, \tau + 2, ..., \tau + \Delta$, as follows:

$$Predict_\omega(\tau, \Delta) = \sqrt[\Delta+1]{\prod_{\delta=0}^{\Delta} Q^*_\omega(\tau, \delta)} \tag{9}$$



White circles are prediction qualities of AML, and red (gray) circles are those of AT. The picture describes how good each model is in predicting the future after having fed data of the first τ months (with $\tau = 12$ – left, and $\tau = 24$ – right).

Fig. 5. The prediction qualities of AML and AT.

where Δ is the prediction time span. We use the geometric mean instead of the arithmetic mean because the temporal quality is a normalized measure [14].

4.5 Step 5: Compare VDMs

This section addresses the second research question RQ2 concerning the comparison between VDMs based on quality and predictability.

VDMs only make sense if they could predict the future trend of vulnerabilities. Hence a VDM which perfectly fits the historical data, but badly estimates the future trend even in a short period, is utterly useless: *a better model is the one that better forecasts the future*.

The comparison between two models vdm_1 and vdm_2 is done as follows. Let ρ_1, ρ_2 be the predictability of vdm_1 and vdm_2, respectively.

$$\begin{aligned} \rho_1 &= \{Predict_{\omega=0.5}(\tau, \Delta) | \tau = 6..\tau_{max}, vdm_1\} \\ \rho_2 &= \{Predict_{\omega=0.5}(\tau, \Delta) | \tau = 6..\tau_{max}, vdm_2\} \end{aligned} \quad (10)$$

where the prediction time span Δ could follow the criteria CR4; $\tau_{max} = \min(72, \max_{r \in R} \tau^r_{max})$. We employ the one-sided Wilcoxon rank-sum test to compare ρ_1, ρ_2. If the returned *p-value* is less than the significance level $\alpha = 0.05$, the predictability of vdm_1 is statistically greater than that of vdm_2. It also means that vdm_1 is better than vdm_2. If *p-value* $\geq 1 - \alpha$, we conclude the opposite *i.e.*, vdm_2 is better than vdm_1. Otherwise we have not enough evidence either way.

If the previous comparison is inconclusive, we repeat the comparison using the value of temporal quality of the VDMs instead of predictability. We just replace $Q_{\omega=0.5}(\tau)$ for $Predict_{\omega=0.5}(\tau, \Delta)$ in the equation (10), and repeat the above activities.

To compare models, we run several hypothesis tests. To avoid the family-wise error rate, which is the probability of making one or more type I errors by chance, we apply the Bonferroni correction: the significance level is divided by the number of tests performed.

Example 12 When we compare one model against other seven models, the Bonferroni-corrected significance level is: $\alpha = {}^{0.05}/_{7} \approx 0.007$. ■

TABLE 6
Vulnerability data sources of browsers.

Data Source	Category	Apply for
National Vulnerability Database (NVD)	TADV	All browsers
Mozilla Foundation Security Advisories (MFSA)	ADV	Firefox
Mozilla Bugzilla (MBug)	BUG	Firefox
Microsoft Security Bulletin (MSB)	ADV	IE
Apple Knowledge Base (AKB)	ADV	Safari
Chrome Issue Tracker (CIT)	BUG	Chrome

The above comparison activities are summarized in the criteria CR5 (see TABLE 4).

5 Vulnerability Data Infrastructure

This section describes the software infrastructure and the data sources used to compute and maintain our vulnerability warehouse. TABLE 6 presents the vulnerability data sources for the browsers in our study.

5.1 Software Infrastructure

Fig. 6 reports the software infrastructure for collecting vulnerability data of browsers. The infrastructure consists of three layers (separated in panels in the figure):

- *Layer 1: Data Collector* includes a Web Crawler that downloads HTML/XML files from the servers of data providers. These files are piped to the HTML/XML Data Extractor to extract interesting data features (Section 4.1). Missing features in some data sources (*e.g.,* version data in MFSA) are obtained by correlating across data sources by the Data Correlator via the refs feature [23].
- *Layer 2: Data Sampler* extracts observed samples ((1) from Section 4.1).
- *Layer 3: Data Analysis* includes the VDM Model Fitting processor and the VDM Quality/Predictability Analysis processor. The former takes the output of Layer 2 and performs model fitting for all data samples to all VDMs. The output is a collection of evaluated samples. The latter processor takes the generated evaluated samples and executes the quality, predictability analysis (see Step 3, Step 4). It also executes the VDMs comparison (Step 5).

To enable the reproducibility of the experiment, we do not apply any *manual* sanitizer to the data collection.

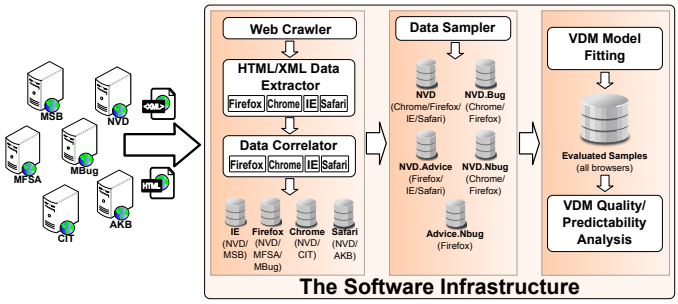
5.2 Collected Vulnerability Data Sets

TABLE 7 reports the descriptive statistics of observed samples in five collected data sets described in TABLE 5. The examined browsers have a very different release cycle. Chrome has a very short cycle (about a month), while other browsers have a longer one in our study. This could explain why the means and standard deviations of Chrome are lower than those of other browsers.

TABLE 7
Descriptive statistics of observed data samples

Column names: med. - median, μ - mean, σ - standard deviation. Dash (–) means data set is not available due to missing data sources.

Browser	Releases	NVD				NVD.Bug				NVD.Advice				NVD.NBug				Advice.NBug				All Data Sets			
		Total	med.	μ	σ	Total	med.	μ	σ	Total	med.	μ	σ	Total	med.	μ	σ	Total	med.	μ	σ	Total	med.	μ	σ
Firefox	8	378	*42*	*47*	*30*	378	*42*	*47*	*30*	378	*42*	*47*	*30*	378	*42*	*47*	*30*	378	*42*	*47*	*30*	1,890	*210*	*236*	*148*
Chrome	12	281	*20*	*23*	*10*	281	*20*	*23*	*10*	–	–	–	–	281	*20*	*23*	*10*	–	–	–	–	843	*62*	*70*	*29*
IE	5	573	*130*	*115*	*59*	–	–	–	–	573	*130*	*115*	*59*	–	–	–	–	–	–	–	–	1,146	*260*	*229*	*118*
Safari	5	314	*60*	*63*	*35*	–	–	–	–	314	*60*	*63*	*35*	–	–	–	–	–	–	–	–	628	*120*	*126*	*69*
Total	30	1,546	*36*	*52*	*44*	659	*27*	*33*	*23*	1,265	*64*	*70*	*48*	659	*27*	*33*	*23*	378	*42*	*47*	*30*	4,507	*104*	*150*	*118*



Round rectangles denote the data preprocessors. Arrow connections indicate the direction of data flows.

Fig. 6. The data processing infrastructure.

The latest time horizon for these data sets is $30th$ June 2012. We selected these major releases because they were at least one year old in the market at the time of the data collection. The rationale is that some VDMs only work for lifetimes of over a a year, as they explicitly try to model the loss of interest by the attacker. For example, Firefox versions 6 through 14 were released before the end date of June 2012, but were not included: not enough data points. In total, we have collected $4,507$ observed samples for 30 major releases of browsers *i.e.*, Chrome v1.0–v12.0, Firefox v1.0–v5.0, IE v4.0–v9.0, and Safari v1.0–v5.0.

6 An Assessment on Existing VDMs

We apply the methodology to assess the performance of eight existing VDMs (see also TABLE 8). In this assessment, we consider 8 out of 10 VDMs listed in the taxonomy (see Fig. 1). Two models MVDM and Effort-based AML (AML-E) are excluded. MVDM by Kim *et al.*[19] require additional data beyond the scope of this study, *i.e.*, the ratio of share code between versions, which is only available for Chrome and Firefox, though Kim *et al.* offered an explanation for the deviation from the logistic shape. AML-E [2] uses the test-effort as the main factor instead of the calendar time, which is not comparable to other models.

We follow Step 2 to fit the above VDMs to collected observed samples. The model fitting relies on the function `nls()` of R [37]. The model fitting took approximately

TABLE 8
The VDMs in evaluation and their equation.

VDMs are listed in the alphabetical order. The meaning of the VDMs' parameters are referred to their original work.

Model	*Equation*
Alhazmi-Malaiya Logistic (AML) [6]	$\Omega(t) = \frac{B}{BCe^{-ABt}+1}$
Anderson Thermodynamic (AT) [8]	$\Omega(t) = \frac{k}{\gamma}\ln(t) + C$
Joh Weibull (JW) [18]	$\Omega(t) = \gamma(1 - e^{-\left(\frac{t}{\beta}\right)^{\alpha}})$
Linear (LN)	$\Omega(t) = At + B$
Logistic Poisson (LP) [26]	$\Omega(t) = \beta_0 \ln(1 + \beta_1 t)$
Rescorla Exponential (RE) [39]	$\Omega(t) = N(1 - e^{-\lambda t})$
Rescorla Quadratic (RQ) [39]	$\Omega(t) = \frac{At^2}{2} + Bt$
Younis Folded (YF) [53]	$\Omega(t) = \frac{\gamma}{2}\left[\mathrm{erf}\left(\frac{t-\tau}{\sqrt{2\sigma}}\right) + \mathrm{erf}\left(\frac{t+\tau}{\sqrt{2\sigma}}\right)\right]$

Note: *erf()* is the error function, $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,dt$

82 minutes on a dual-core 2.73GHz Windows machine with 6GB of RAM yielding $31,241$ curves in total. During the model fitting, `nls()` is unable to fit some models in some observed samples. Hence the number of generated curves are less than the number of observed samples multiplied by the number of VDMs.

TABLE 9 reports the number of evaluated samples for each VDM in each data set. We also report the percentage of *Good Fit*, *Inconclusive Fit*, and *Not Fit* in each data set. Apparently, AML and YF obtain more *Good Fits* than other models, in relative percentage of the number of evaluated samples in each data set. Additionally, VDMs obtain more *Good Fits* in NVD.Advice than other data sets.

6.1 Goodness-of-Fit Analysis for VDMs

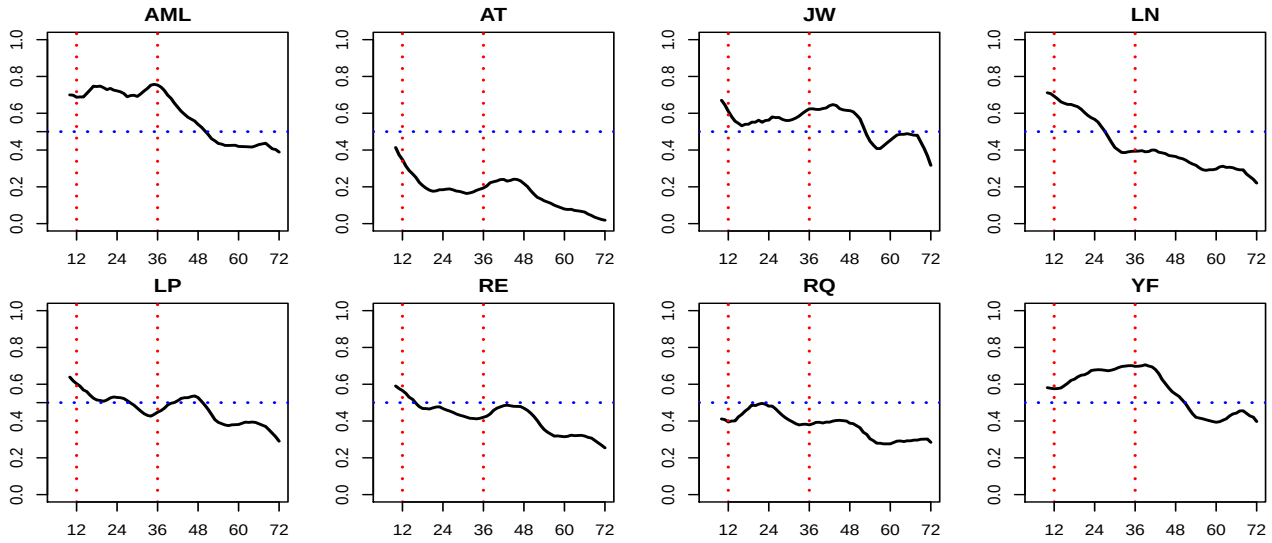
The analysis is conducted on all evaluated samples from all collected data sets. The inconclusiveness contribution factor ω is set to 0.5 as described in CR3. We reuse the three-phase idea from the AML model to divide the lifetime of a browser into three periods: *young* – when a browser has been released for 12 months or less; *middle-age* – released for 13 – 36 months; and *old* – released more than 36 months.

Fig. 7 exhibits the moving average of temporal quality $Q_\omega(\tau)$. We cut $Q_\omega(\tau)$ at horizon 72 though we have

TABLE 9
Overall distribution of evaluated samples.

Column names: G.F - Good Fit, I.F - Inconclusive Fit, N.F - Not Fit.

Model	NVD				NVD.Bug				NVD.Advice				NVD.NBug				Advice.NBug				All Data Sets			
	Total	G.F	I.F	N.F	Total	G.F	I.F	N.F	Total	G.F	I.F	N.F	Total	G.F	I.F	N.F	Total	G.F	I.F	N.F	Total	G.F	I.F	N.F
AML	1,375	**43%**	32%	24%	559	30%	**48%**	22%	1,064	**49%**	24%	27%	559	**77%**	12%	10%	338	**62%**	13%	25%	3,895	**49%**	28%	23%
AT	1,378	8%	18%	**74%**	559	10%	15%	**75%**	1,157	9%	17%	**74%**	559	8%	15%	**77%**	338	5%	38%	**57%**	3,991	8%	19%	**73%**
JW	1,344	39%	18%	**44%**	547	28%	30%	**42%**	1,019	**64%**	10%	26%	551	40%	14%	**46%**	336	**60%**	21%	19%	3,797	**46%**	17%	37%
LN	1,378	36%	19%	**45%**	559	20%	31%	**49%**	1,157	41%	16%	**43%**	559	**41%**	23%	36%	338	40%	15%	**45%**	3,991	36%	20%	**44%**
LP	1,377	42%	14%	**43%**	559	19%	34%	**46%**	1,069	**46%**	13%	41%	559	28%	20%	**52%**	338	33%	**46%**	20%	3,902	37%	20%	**42%**
RE	1,378	41%	14%	**44%**	559	20%	34%	**46%**	1,069	**46%**	13%	41%	559	13%	27%	**60%**	338	17%	30%	**52%**	3,903	33%	20%	**47%**
RQ	1,378	29%	20%	**51%**	559	24%	34%	**43%**	1,157	**50%**	10%	39%	559	14%	13%	**74%**	338	4%	2%	**94%**	3,991	30%	16%	**53%**
YF	1,358	**55%**	20%	25%	551	**54%**	29%	17%	966	**71%**	11%	19%	558	28%	22%	**50%**	338	14%	7%	**78%**	3,771	**51%**	18%	31%
Total	10,966	37%	19%	**44%**	4,452	26%	32%	**43%**	8,658	**46%**	14%	40%	4,463	31%	18%	**51%**	2,702	30%	22%	**49%**	31,241	36%	20%	**44%**



The X-axis is the number of months since release (*i.e.,* horizon τ). The Y-axis is the value of temporal quality. The solid lines are the moving average of $Q_{\omega=0.5}(\tau)$ with window size $k = 5$. The dotted horizontal line at 0.5 is the base line to assess VDM. Vertical lines are the marks of the horizons of $12th$ and $36th$ month.

Fig. 7. The trend of temporal quality $Q_{\omega=0.5}(\tau)$ of the VDMs in first 72 months.

more data for some systems (*e.g.*, IE v4, FF v1.0): the vulnerability data reported for versions released after 6 years might be not reliable, and might overfit the VDMs.

Fig. 7 shows a clear evidence that both AT and RQ should be rejected since their temporal qualities always sink below the base line (*i.e.*, less than 0.5). Other models may be adequate when browsers are young. AML and LN look better than other models in this respect.

In the middle-age period, AML is still relatively good. JW and YF improve when approaching month $36th$ though JW get worse after month $12th$. The quality of both LN and LP worsen after month $12th$, and sink below the base line when approaching month $36th$. RE is almost below the base line after month $15th$. Hence, in the middle-age period, AML, JW, and YF models may be adequate; LN and LP are deteriorating but might be still considered adequate; RE should clearly be rejected.

When browsers are old (36+ months), AML, JW, and YF deteriorate and dip below the base line since month $48th$ (approx.), while others collapse since month $36th$. Additional boxplot comparisons can be found in [30].

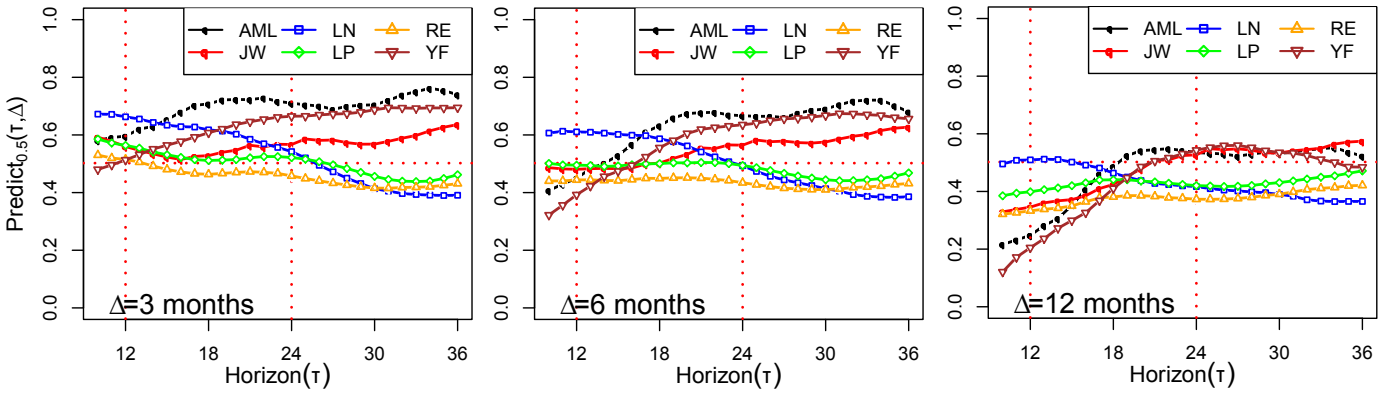
In summary, our quality analysis shows that:

- AT and RQ models should be rejected.
- All other models may be adequate when browser is young. Only s-shape models (*i.e.*, AML, YW, YF) might be adequate when browsers are middle-age.
- No model is good enough when browsers are old.

6.2 Predictability Analysis for VDMs

From the previous quality analysis, AT and RQ models are low quality. Hence, we exclude these models from the predictability analysis. Furthermore, since no model is good when browsers are too old, we analyze the predictability of these models only for the first 36 months from a release date. This period is still large as recent releases live less than a year [12], [44], [45], [46].

Predictability is a bi-dimensional function as it takes the horizon of data collection for fitting and the predic-



A horizonal line at value of 0.5 is the base line to assess the predictability. LN's predictability is above the base line (*i.e.,* adequate) til month $24th$ for a relative short prediction time span (3-6 months). The predictability of S-shape models is adequate after month $12th$ for a relative short time span (3-6 months), after month $18th$ for longer time span (12 months).

Fig. 8. The predictability of VDMs at fixed prediction time spans Δ for varying horizons τ.

tion time. Fig. 5 shows a graph where the horizon is fixed at 12 and 24 while the prediction time varies and the ability to predict invariably decreases as we move further into the future. Here we keep the prediction time Δ fixed and let the fitting horizon τ vary: our purpose is to understand which is the best model for a given time horizon, see Fig. 8.

Fig. 8 reports the moving average VDMs' predictability along horizons in different prediction time spans. The horizonal line at 0.5 is the base line for a qualitative assessment of the predictability of VDMs (as same as the temporal quality of VDMs). The predictability lines go down (model is good at the beginning but deteriorates with software ages) as well as up (model is more appropriate for older software).

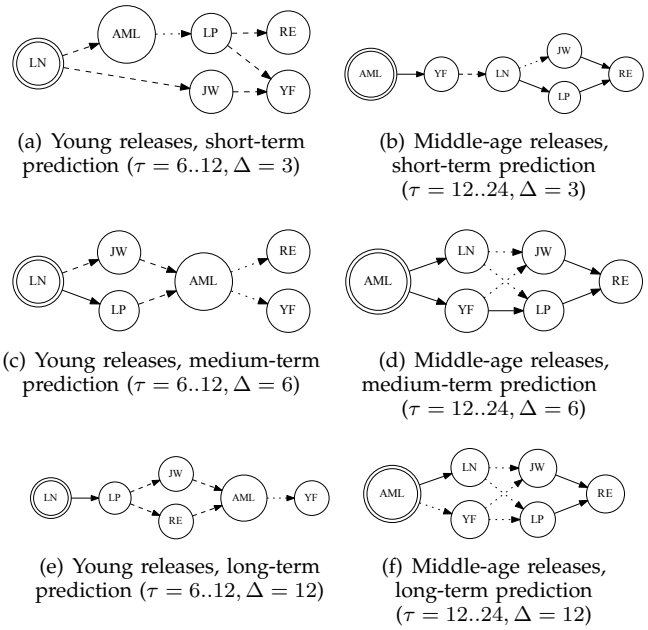
For the first year after the release date of a major version ($\tau \leq 12$), the predictability of LN is the best for all prediction time spans ($\Delta = 3, 6, 12$). All other models are under performing and well below the LN line. At around $15 - 18$ months, the AML predictability line overtakes the LN line. S-shape models (AML, JW, and YF) are inadequate for young software, but improves as software ages. They become adequate after month $18th$ and keep being so until the end of the study period. The LP and RE models are usually below the others and below the base line except when the browser is young ($\tau < 12$), and the prediction time span is short ($\Delta = 3$).

When the prediction time span is very long (*i.e.,* 24 months) no model is good enough as all models sink below the base line.

6.3 Comparison of VDMs

The comparison between VDMs follows Step 5. Instead of reporting tables of *p-value*s, we visualize the comparison results in terms of directed graphs where nodes represent models, and connections represent the order relationship between models.

Fig. 9 summarizes comparison results between models for different horizons (τ) and prediction time spans (Δ)



(a) Young releases, short-term prediction ($\tau = 6..12, \Delta = 3$)

(b) Middle-age releases, short-term prediction ($\tau = 12..24, \Delta = 3$)

(c) Young releases, medium-term prediction ($\tau = 6..12, \Delta = 6$)

(d) Middle-age releases, medium-term prediction ($\tau = 12..24, \Delta = 6$)

(e) Young releases, long-term prediction ($\tau = 6..12, \Delta = 12$)

(f) Middle-age releases, long-term prediction ($\tau = 12..24, \Delta = 12$)

A directed connection from two nodes determines that the source model is better than the target one with respect to their predictability (dashed line), or their quality (dotted line), or both (solid line). A double circle marks the best model. RQ and AT are not shown as they are the worst models. LN is the best model to predict trend of vulnerabilities for young browsers, otherwise AML is the best model for middle-age browsers.

Fig. 9. Comparison results among VDMs.

along the following convention:

- *Solid line*: predictability and quality of the source is significantly better than the target's.
- *Dashed line*: predictability of the source is significantly better than the target.
- *Dotted line*: quality of the source is significantly better than the target.

By the word *significantly*, we means the *p-value* of the corresponding one-sided Wilcoxon rank-sum test is less than the significance level. We apply the Bonferroni correction to control the multi comparison problem, hence

TABLE 10
Suggested models for different usage scenarios.

Observation Period (month)	Prediction Time Span (month)		Best Model	2nd Best Model(s)
6 – 12	3	(short-term)	LN	AML, JW
6 – 12	6	(medium-term)	LN	JW, LP
6 – 12	12	(long-term)	LN	LP
13 – 24	3	(short-term)	AML	YF
13 – 24	6	(medium-term)	AML	YF, LN
13 – 24	12	(long-term)	AML	YF, LN

TABLE 11
A potentially misleading results of fitting VDMs in the largest horizon of browser releases, using NVD data sets

The goodness of fit of a VDM is based on *p-value* in the χ^2 test. *p-value* < 0.05: not fit (×), *p-value* ≥ 0.80: good fit (✓), and inconclusive fit (blank) otherwise. It is calculated over the entire lifetime.

	Firefox							Chrome											IE				Safari				
	1	1.5	2	3	3.5	3.6	4	5	1	2	3	4	5	6	7	8	9	10	11	12	4	5	6	7	8	1	2	3	4	5
AML	×	×		×		✓			✓		×				×		✓			×	✓		×		✓	×				✓
AT	×	×	×	×	×	×			×	×	×	×	×	×	×	×	×	×	×		×	×	×	×	×	×	×	×	×	
JW	×	×	✓	✓	✓	✓	✓		×	×	×	×			✓	✓					×	✓	×	✓		×	×	×	×	×
LN	×	×		×			✓		×	×	×	×								×	×	×	×	✓		×	×	×	×	
LP	×	×	✓	✓	✓	✓	✓		×	×	×	×	×				×				×	✓	×	×	✓	×	×	×	×	
RE	×	×	✓	✓	✓	✓	✓		×	×	×	×	×				×				×	✓	×	×	✓	×	×	×	×	
RQ	×	×	×	×	×		✓		×	×	×					×	×	×	×	×	×	×	×	×	✓	×	×	×	×	×
YF	×	×	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓		×				✓	✓	✓	×	✓	×	×	×	×	

the significance level is: $\alpha = 0.05/5 = 0.01$.

Based on Fig. 9, TABLE 10 suggests model(s) for different usage scenarios described in CR4 (see TABLE 4).

In short, *when browsers are young, the LN model is the most appropriate choice. This is because the vulnerability discovery process is linear. When browsers are approaching middle-age, the AML model becomes superior*.

7 Discussion

This section compares our methodology to the traditional evaluation method described in Section 3. For the traditional methodology: VDMs are fitted to the NVD data set at the largest horizon. In other words, we use following observed samples to evaluate VDMs:

$$OS_{\mathsf{NVD}} = \{\mathsf{TS}(r, \mathsf{NVD}, \tau^r_{max}) | r \in R\}$$

where R is the set of all releases mentioned in Section 5.2.

The fitting results are reported in TABLE 11. To improve readability, we report the categorized goodness-of-fit based on the *p-value* (see CR2) instead of the raw *p-value*s. As in the literature we set the *inconclusiveness contribution* factor $\omega = 1$.

The table shows that two models AT and RQ have a large number of *Not Fits* entries (90% and 70% respectively); whereas other models have less than 50% *Not Fits* entries. Only the YF model has more than 50% *Good Fits* entries. Some systems have long gone into retirement. For example, FF v2.0 vulnerabilities are no longer sought by researchers. The publication of those vulnerabilities is a byproduct of research on later versions.

From TABLE 11, we might conclude: *(1)* AML and YF are the "best" model (YF is slightly better than AML); *(2)* AT and RQ are the "worst"; and *(3)* other models are approximately equal.

With reference to TABLE 1 (or TABLE 10), the conclusions obtained by the traditional methodology are clearly less informative than those by our methodology. They both agree that AT and RQ are the worst. However, our methodology provides statistical evidences about the superior performance of LN and AML in *different* periods of browser lifetime, whereas the traditional one does not.

An interesting issue is whether we need the full complexity of the methodology and could attain the same insight by a reduced number of steps. We could have stopped the analysis at Step 2, after TABLE 9. This table does not distinguish between AML and YF. Moreover, it obfuscates the greater performance of LN for the first year since release. This is a major issue for today's software whose lifecycle is very short.

In summary, our methodology provides more practical information about the performance of VDMs in different scenarios than the traditional methodology.

8 Threats to Validity

Some threats to **Construct validity** may affects our data.

Bugs in data collector. The vulnerability data is collected by a crawler parsing HTML pages. It might be buggy. We minimize this threat by randomly checking the data. When an error was found we corrected the crawler and recollected the data.

Bias in bug-to-nvd linking scheme. While collecting data for Advice.Nbug, we apply heuristics to link a bug to an NVD entry based on their positions in an MFSA report. We manually checked many links for inconsistencies.

Bias in bug-affects-version identification. We do not completely known which bugs affect which versions. We assume that a bug affects all versions mentioned in its linked NVD entry. This might overestimate the number of bugs in each version. So, we estimate the latest release that a bug might impact, and filter all vulnerable releases after this latest. Such estimation is done by the technique discussed in [43], [31]. The potential errors in NVD discussed in [31] only affect the retrospective fitness of models over the long term so only valuations after 36 months might be affected.

Internal validity threats affect the causal relationship between the collected data and the conclusion in the study. Our conclusions are based on statistical tests. We analyzed the tests assumptions to make sure no unwarranted assumption was present. We did not apply any tests with normality assumptions since the distribution of vulnerabilities is not normal.

Notably, the collected data sets in TABLE 7 are not independent each others, for instance NVD.Bug and NVD.Advice are sub sets of NVD. They represent for different ways of counting vulnerabilities from the NVD

data source. We treat these data sets equally and fit them all to VDMs to address the issues behind *vulnerability count methods* (Section 3). This might have an accumulative effects on quality and predictability. We believe this accumulative effect is negligible because TABLE 9, reporting statistics of evaluated samples, shows that responses of VDMs to different data sets are very different.

Another potential internal validity problem is the 'independently distributed' assumption that statistical tests might violate. This could be the case when developers discover a vulnerability, realize they make mistake elsewhere, and go back to fix other mistakes in the same area of the system. For reported vulnerabilities in Firefox and Chrome, the analysis of the commit logs shows that each fix mostly corresponds to a fix of a single vulnerability, and each fix touches very few components (on average, 1.17 for Firefox, and 1.84 for Chrome [30]). It means the problem is minor in our study, at least for Firefox and Chrome. If mistakes are not reported as vulnerability entries, they do not "exists" for us, and therefore the reported entries are indeed independent (the dependent events are not present at all).

External validity is the extent to which our conclusion could be generalized to other scenarios. Our experiment is based on the four most popular browsers. So we can be confident about our conclusion for browsers in general, but it might not be valid for other types of application such as operating systems. Such validity requires additional experiments.

9 Conclusion

Vulnerability discovery models (VDMs) have the potential to help us in predicting future trends of vulnerabilities, adapting software update and patching schedule, or selecting (open source) products to bundle into other proprietary products.

The major contribution of this work is an empirical methodology to conduct evaluation experiments on VDMs. The quality and predictability of VDMs in question are measured by two functions that report the ability to fit vulnerability data and the ability to predict future vulnerabilities as function of time. As a result, we obtain a better insight about VDMs.

To illustrate the methodology, the paper reports an evaluation experiment to assess eight VDMs (*i.e.*, AML, AT, LN, JW, LP, RE, RQ, and YF) on 30 major releases of four web browsers: IE, Firefox, Chrome, and Safari. We classify the age of a browser's version in three different periods: youth (within $6-12$ months since release date), middle age ($12-36$ months since release date), and old age (beyond 36 months). Our experiment reveals interesting findings on these existing VDMs: if a version is relatively young, then we should use a linear model to estimate the vulnerabilities in the next $3-6$ months. For middle-aged browsers it is better to use an s-shape logistic model. This would have insight been impossible to achieve with the traditional methodology.

A number of further investigations is possible such as replicate the experiment with other software types. We did not consider MVDM model [19] and effort-based models because they require data that is beyond the scope of this study. An interesting issue is understanding whether the speed of the release cycle and/or the size of the releases impacts VDMs quality. Another question is evaluating how errors in the datasets may impact the analysis [31]. Further assessments in these directions will make the understanding of VDMs more comprehensive.

10 Replication Guideline

We have made the data in this work available online. Interested readers who want to reproduce the experiment could attain the data at http://securitylab.disi.unitn.it/.

Acknowledgements

We would like to thank all anonymous reviewers for their helpful comments and suggestion. This work has been partly supported by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 285223 - SECONOMICS, and the Italian Project MIUR-PRIN-TENACE.

References

[1] H. Akaike. Prediction and entropy. In A. C. Atkinson and S. E. Fienberg, editors, *A Celebration of Statistics*, pages 1–24. Springer New York, 1985.

[2] O. Alhazmi and Y. Malaiya. Modeling the vulnerability discovery process. In *Proc. of the 16th IEEE Internat. Symp. on Software Reliability Engineering (ISSRE'05)*, pages 129–138, 2005.

[3] O. Alhazmi and Y. Malaiya. Measuring and enhancing prediction capabilities of vulnerability discovery models for Apache and IIS HTTP servers. In *Proc. of the 17th IEEE Internat. Symp. on Software Reliability Engineering (ISSRE'06)*, pages 343–352, 2006.

[4] O. Alhazmi and Y. Malaiya. Prediction capabilities of vulnerability discovery models. In *Proc. of the Reliability and Maintainability Symp. (RAMS'06)*, pages 86–91, 2006.

[5] O. Alhazmi and Y. Malaiya. Application of vulnerability discovery models to major operating systems. *IEEE Transactions on Reliability*, 57(1):14–22, 2008.

[6] O. Alhazmi, Y. Malaiya, and I. Ray. Security vulnerabilities in software systems: A quantitative perspective. In S. Jajodia and D. Wijesekera, editors, *Data and Applications Security XIX*, volume 3654 of *LNCS*, pages 281–294. Springer, 2005.

[7] O. Alhazmi, Y. Malaiya, and I. Ray. Measuring, analyzing and predicting security vulnerabilities in software systems. *Computer & Security*, 26(3):219–228, 2007.

[8] R. Anderson. Security in open versus closed systems - the dance of Boltzmann, Coase and Moore. In *Proc. of Open Source Software: Economics, Law and Policy*, 2002.

[9] W. A. Arbaugh, W. L. Fithen, and J. McHugh. Windows of vulnerability: A case study analysis. *IEEE Computer*, 33(12):52–59, 2000.

[10] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[11] I. Chowdhury and M. Zulkernine. Using complexity, coupling, and cohesion metrics as early predictors of vulnerabilities. *Journal of System Architecture*, 57(3):294–313, 2011.

[12] Chromium Developers. Chrome stable releases history, 2012. http://omahaproxy.appspot.com/history?channel=stable, visited in July 2012.

[13] M. Dowd, J. McDonald, and J. Schuh. The art of software security assessment. Addision-Wesley publications, 2007.

[14] P. J. Fleming and J. J. Wallace. How not to lie with statistics: the correct way to summarize benchmark results. *Communication of the ACM*, 29(3):218–221, 1986.

[15] M. Gegick. Failure-prone components are also attack-prone components. In *OOPSLA - ACM student research competition*, pages 917–918, 2008.

[16] M. Gegick, P. Rotella, and L. Williams. Toward non-security failures as a predictor of security faults and failures. In *Proc. of the 2009 Engineering Secure Software and Systems Symp. (ESSoS'09)*, volume 5429, pages 135–149, 2009.

[17] A. L. Goel and K. Okumoto. A time dependent error detection model for software reliability and other performance measures. *IEEE Transactions on Reliability*, R-28:206–211, 1979.

[18] H. Joh, J. Kim, and Y. Malaiya. Vulnerability discovery modeling using Weibull distribution. In *Proc. of the 19th IEEE Internat. Symp. on Software Reliability Engineering (ISSRE'08)*, pages 299–300, 2008.

[19] J. Kim, Y. Malaiya, and I. Ray. Vulnerability discovery in multi-version software systems. In *Proc. of the 10th IEEE Internat. Symp. on High Assurance Systems Engineering*, pages 141–148, 2007.

[20] I. V. Krsul. *Software Vulnerability Analysis*. PhD thesis, Purdue University, 1998.

[21] Y. K. Malaiya, N. Karunanithi, and P. Yerma. Predictability of software reliability models. *IEEE Transactions on Reliability*, 41(4):539–546, 1992.

[22] F. Massacci, S. Neuhaus, and V. H. Nguyen. After-life vulnerabilities: A study on firefox evolution, its vulnerabilities and fixes. In *Proc. of the 2011 Symp. on Engineering Secure Software and Systems (ESSoS'11)*, 2011.

[23] F. Massacci and V. H. Nguyen. Which is the right source for vulnerabilities studies? an empirical analysis on mozilla firefox. In *Proc. of the Internat. ACM Workshop on Security Measurement and Metrics (MetriSec'10)*, 2010.

[24] S. McKillup. *Statistics Explained: An Introductory Guide for Life Scientists*. Cambridge University Press, 2005.

[25] C. Miller. The legitimate vulnerability market: Inside the secretive world of 0-day exploit sales. In *Proc. of the 6th Workshop on Economics and Information Security*, 2007.

[26] J. D. Musa and K. Okumoto. A logarithmic poisson execution time model for software reliability measurement. In *Proc. of the 7th Internat. Conf. on Software Engineering*, ICSE '84, pages 230–238, Piscataway, NJ, USA, 1984. IEEE Press.

[27] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *Proc. of the 26th Internat. Conf. on Software Engineering*, pages 284–292, 2005.

[28] R. Needham. Security and open source. In *Open Source Software Economics*, 2002. Available at http://idei.fr/doc/conf/sic/papers_2002/needham.pdf.

[29] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller. Predicting vulnerable software components. In *Proc. of the 14th ACM Conf. on Computer and Comm. Security (CCS'07)*, pages 529–540, 2007.

[30] V. H. Nguyen. *Empirical Methods for Evaluating Empirical Vulnerability Models*. PhD thesis, University of Trento, 2014.

[31] V. H. Nguyen and F. Massacci. The (un) reliability of nvd vulnerable versions data: an empirical experiment on google chrome vulnerabilities. In *Proc. of the 8th ACM Symp. on Information, Computer and Comm. Security (ASIACCS'13)*, 2013.

[32] V. H. Nguyen and L. M. S. Tran. Predicting vulnerable software components using dependency graphs. In *Proc. of the Internat. ACM Workshop on Security Measurement and Metrics (MetriSec'10)*, 2010.

[33] NIST. *e-Handbook of Statistical Methods*, 2012. http://www.itl.nist.gov/div898/handbook/.

[34] H. M. Olague, S. Gholston, and S. Quattlebaum. Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. *IEEE Transactions on Software Engineering*, 33(6):402–419, 2007.

[35] A. Ozment. Improving vulnerability discovery models: Problems with definitions and assumptions. In *Proc. of the 3rd Workshop on Quality of Protection*, 2007.

[36] A. Ozment and S. E. Schechter. Milk or wine: Does software security improve with age? In *Proc. of the 15th USENIX Security Symp.*, 2006.

[37] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0.

[38] M. Rajab, L. Ballard, N. Jagpal, P. Mavrommatis, D. Nojiri, N. Provos, and L. Schmidt. Trends in circumventing web-malware detection. Technical report, Google, 2011.

[39] E. Rescorla. Is finding security holes a good idea? *IEEE Security and Privacy*, 3(1):14–19, 2005.

[40] F. B. Schneider. Trust in cyberspace. *National Academy Press*, 1991.

[41] G. Schryen. Security of open source and closed source software: An empirical comparison of published vulnerabilities. In *Proc. of 15th Americas Conf. on Information Systems (AMCIS'09)*, 2009.

[42] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. *IEEE Transactions on Software Engineering*, 37(6):772–787, 2011.

[43] J. Sliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *Proc. of the 2nd Internat. Working Conf. on Mining Software Repositories MSR('05)*, pages 24–28, May 2005.

[44] Wikipedia. Firefox release history, 2012. http://en.wikipedia.org/wiki/Firefox_release_history, visited in July 2012.

[45] Wikipedia. Internet explorer, 2012. http://en.wikipedia.org/wiki/Internet_Explorer, visited in July 2012.

[46] Wikipedia. Safari version history, 2012. http://en.wikipedia.org/wiki/Safari_version_history, visited in July 2012.

[47] S.-W. Woo, O. Alhazmi, and Y. Malaiya. An analysis of the vulnerability discovery process in web browsers. In *Proc. of the 10th IASTED Internat. Conf. on Software Engineering and Applications*, 2006.

[48] S.-W. Woo, O. Alhazmi, and Y. Malaiya. Assessing vulnerabilities in Apache and IIS HTTP servers. In *Proc. of the 2nd IEEE Internat. Symp. on Dependable, Autonomic and Secure Computing*, 2006.

[49] S.-W. Woo, H. Joh, O. Alhazmi, and Y. Malaiya. Modeling vulnerability discovery process in Apache and IIS HTTP servers. *Computer & Security*, 30(1):50 – 62, 2011.

[50] A. Wood. Software reliability growth models. Technical report, Tandem Computer, 1996.

[51] S. Yamada, M. Ohba, and S. Osaki. S-shaped reliability growth modeling for software error detection. *IEEE Transactions on Reliability*, R-32:475–484, 1983.

[52] Y. Younan. 25 years of vulnerabilities:1988-2012. Technical report, Source Fire, 2013.

[53] A. Younis, H. Joh, and Y. Malaiya. Modeling learningless vulnerability discovery using a folded distribution. In *Proc. of the Internat. Conf. Security and Management (SAM'11)*, pages 617–623, 2011.

[54] T. Zimmermann and N. Nagappan. Predicting subsystem defects using dependency graph complexities. In *Proc. of the 20th IEEE Internat. Symp. on Software Reliability Engineering (ISSRE'09)*, 2007.

Fabio Massacci is a full professor at the University of Trento. He has a Ph.D. in Computing from the University of Rome La Sapienza in 1998. He has been in Cambridge (UK), Toulouse (FR) and Siena (IT). Since 2001 he is Trento. He has published more than 100 articles on security and his current research interest is in empirical methods for security. He is the European Coordinator of the multi-disciplinary research project SECONOMICS on socio-economic aspects of security. Contact him at Fabio.Massacci@unitn.it

Viet Hung Nguyen obtained his Ph.D. in ICT at University of Trento (UNITN) in 2014, under the supervision of professor Fabio Massacci. He was a software engineer in the software development industry before joining in UNITN. Currently, his main interests include software security, vulnerability analysis and prediction. Contact him at viethung.nguyen@unitn.it