Predicting the Discovery Pattern of Publically Known Exploited Vulnerabilities

Yazdan Movahedi, Michel Cukier, and Ilir Gashi

Abstract— Vulnerabilities with publically known exploits typically form 2-7% of all vulnerabilities reported for a given software version. With a smaller number of known exploited vulnerabilities compared with the total number of vulnerabilities, it is more difficult to model and predict when a vulnerability with a known exploit will be reported. In this paper, we introduce an approach for predicting the discovery pattern of publically known exploited vulnerabilities using all publically known vulnerabilities reported for a given software. Eight commonly used vulnerability discovery models (VDMs) and one neural network model (NNM) were utilized to evaluate the prediction capability of our approach. We compared their predictions results with the scenario when only exploited vulnerabilities were used for prediction. Our results show that, in terms of prediction accuracy, out of eight software we analyzed, our approach led to more accurate results in seven cases. Only in one case, the accuracy of our approach was worse by 1.6%.

Index Terms— Prediction, Exploited Vulnerabilities, All Vulnerabilities, Vulnerability Discovery Models, Artificial Neural Network, Time to Next Vulnerability

1 Introduction

Researchers have used data from various vulnerability databases to study trends of discovery of new vulnerabilities, used various models for fitting the vulnerability discovery process, and predicting the number of new vulnerabilities that may be discovered for a given product [1]–[6]. Estimating the number of new vulnerabilities over time is useful both for vendors of these products as well as the end-users as it can help them with resource allocation.

For some vulnerabilities, exploits are never published. This might be because the patches for these vulnerabilities are made available very quickly by the vendors, and hence it is not profitable for hackers to develop exploits for them; the vulnerabilities have a lower criticality from the security viewpoint; or it might be that the exploits for these vulnerabilities are only known to the vendors, to security agencies or are exchanged in, for example, dark web forums. Previous studies [7], [8] have reported that vulnerabilities with publically-known exploits usually form only 2-7% of all vulnerabilities reported for a given software version . In addition, as opposed to vulnerability databases such as NVD, which are actively maintained, security repositories reporting exploited vulnerabilities like Exploit Database, also known as "ExploitDB", are less common. A comparison between NVD and ExploitDB finds that only 22% of NVD distinct vulnerabilities have exploits listed in ExploitDB. On the other hand, vulnerabilities with known exploits are more dangerous to end users, even if patches may be available, since not all users regularly patch their systems. For this reason, it is important for both vendors and users to be able to predict the time to the next vulnerability with a known exploit and the number of vulnerabilities that will be exploited over time. However, with a smaller number of known exploited vulnerabilities compared with the total number of vulnerabilities, it is difficult to model and predict the discovery pattern of publically known exploited vulnerabilities. Specifically, the data scarcity makes it difficult to use data driven models, which are helpful where there is no theoretical guidance to explain the data generation process for such data [9]. Therefore, we postulate that it is a worthwhile research activity to explore whether there is a link between discovery pattern of all vulnerabilities reported for a given software and discovery pattern of its exploited vulnerabilities. Finding such link would allow to use a larger dataset of all vulnerabilities for predicting the number of exploited vulnerabilities that will be reported over time.

In this paper, we introduce an approach for predicting the discovery pattern of publically known exploited vulnerabilities using all vulnerabilities reported for a given software. Eight commonly used vulnerability discovery models (VDMs) as well as one neural network model (NNM) were used to evaluate the prediction capability of our approach. We applied the models to vulnerability data associated with four well-known operating systems (OSs) (Window**s**, Mac, IOS (the OS associated with Cisco), and Linux), as well as four well-known web browsers (Internet Explorer, Safari, Firefox, and Chrome).

Two scenarios were considered. In the first scenario (S1), for each software, we utilize all vulnerabilities reported for it (exploited + unexploited) to predict the discovery pattern of exploited vulnerabilities over time. In the second scenario (S2), for each software, we only use exploited vulnerabilities to predict the discovery pattern of exploited vulnerabilities over time.

Our work makes the following contributions:

• *Yazdan Movahedi is with the Center for Risk and Reliability, University of Maryland, College Park, MD 20742. E-mail: ymovahed@ umd.edu.*
• *Michel Cukier is with the Center for Risk and Reliability, University of Maryland, College Park, MD 20742. E-mail: mcukier@ umd.edu.*
• *Ilir Gashi is with the Center for Software Reliability, City, University of London, London, UK EC1V 0HB. E-mail: ilir.gashi.1@city.ac.uk.*

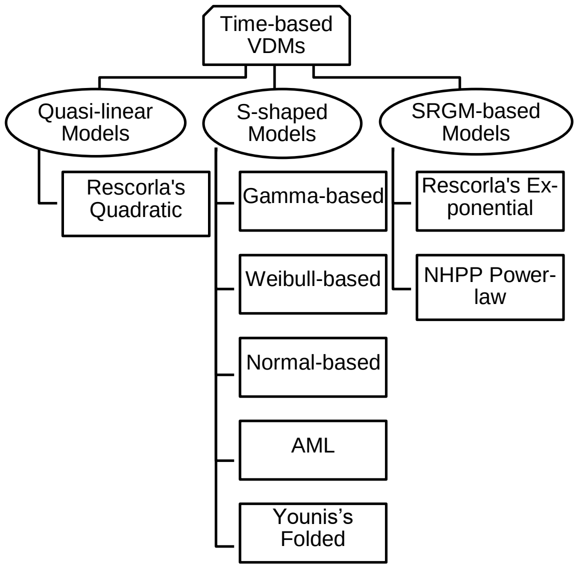


Fig. 1. Classification of Considered Time-based VDMs

- We introduce an approach for predicting total number of publically-known exploited vulnerabilities using all vulnerabilities reported for a given software in 30-day time intervals;
- We compare the prediction capability of two scenarios S1 and S2, S1 when all the vulnerabilities are considered, S2 when only exploited vulnerabilities are, utilizing eight VDMs and one NNM on eight well-known software;
- We show that, out of eight software we analyzed, scenario S1 outperforms scenario S2 in seven cases in terms of prediction accuracy. Only in one case, the prediction of S1 was worse than S2 by 1.5%. In other words, for most of the cases analyzed, we show that using all the vulnerability data available for a system allows to better predict when vulnerabilities that will have publically known exploits for them will be reported.

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 describes the models used in this study, including details of a neural network model we used in our analysis. Section 4 describes the dataset and the scenarios we used. Sections 5 describes the analytical steps we followed in the first scenario (S1). Section 6 presents the results of using both scenarios with the models for prediction. Section 7 discusses the main findings and some limitations. Finally, Section 8 presents conclusions and provisions for future work.

2 Related Work

A security vulnerability is defined as any fault in a software that, if exploited, can lead to a security failure. Research has been conducted to find a link between the fault discovery process of a software and the discovery process of its vulnerabilities for modeling purposes [10]. When considering the fault detection process of a software, it is justifiable to conclude that software reliability models (SRMs) and vulnerability discovery models (VDMs) are similar [1]. In such cases, the intensity/rate function can represent the detection rate of vulnerabilities. Several studies have been conducted applying regression models and/or existing VDMs/SRMs on vulnerability datasets, or proposing new VDMs for modeling the discovery process of vulnerabilities. In these cases, researchers introduced new software security indicators such as total number of residual vulnerabilities in the system, time to next vulnerability (TTNV), vulnerability detection rate, etc. [1]–[7], [10]–[14].

The earliest effort at modeling software reliability was a Markov birth-death model introduced by Hudson in 1967 [15]. A comprehensive overview of several SRMs that characterize the process of software defect-finding is provided in [2]. The earliest study on modeling the vulnerability discovery process was conducted in 2002, when the first VDM termed the Anderson Thermodynamic (AT) model proposed by Anderson [16]. Since 2002, other VDMs have been proposed. Rescorla [4], [5] proposed a VDM to estimate the number of undiscovered vulnerabilities. In 2005, Alhazmi et al. [17] proposed the application of SRMs to vulnerability discover modeling. The same year, they also introduced a logistic VDM known as Alhazmi–Malaiya Logistic (AML) model. Their proposed AML model assumes a symmetrical shape around the peak discovery rate value [6]. A Weibull distribution-based VDM was proposed by Kim in 2007 [18]. Li et al. [19] empirically showed that, in comparison to other reliability models, a Weibull model is better for defect occurrence across a wide range of software systems.

Several studies applied existing models to different types of software packages, such as operating systems and web servers, to simulate the vulnerability discovery rate and predict the number of vulnerabilities that may be present but not yet found [20]–[22]. Other studies focused on increasing the accuracy of vulnerability discovery modeling by examining the skewness of the vulnerability data [23], using Bayesian theorem [24], [25], or using machine learning techniques like nueral networks [26].

In addition to the vulnerabilities publication dates, some studies used software source code for vulnerability assessment in the context of VDMs. Kim et al. [18] proposed a VDM based on shared source code measurements among multi-version software systems. In 2006, Ozment and Schechter employed a reliability growth model to evaluate the security of the OpenBSD OS by examining its source code and the rate at which new code has been introduced [27]. However, it has been shown that source code cannot be an efficient measure in terms of prediction [3]. Recently, Nguyen et al. proposed an automated method that determines the code evidence for the presence of vulnerabilities in previous software versions to evaluate whether the target version is vulnerable [28].

There is little work that focuses on specifically modeling exploited vulnerabilities. One effort is the probabilistic examination of intrusions by [29], [30]. The lack of data is a barrier to modeling exploited vulnerabilities using current VDMs or machine learning techniques, which require considerable amount of data for satisfactory training.

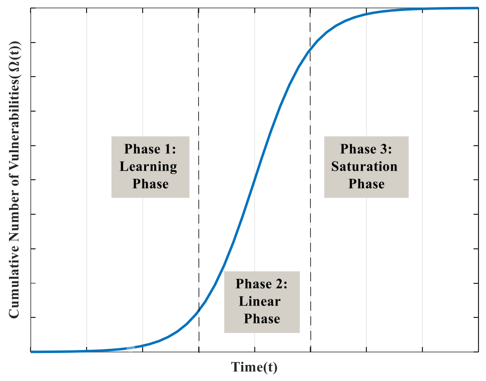


Fig. 2. Three Phases for S-shaped Models

TABLE 1
TABLE OF MODELS AND THEIR EQUATIONS

Model	Equation
NHPP Power-law [32]	$\Omega(t) = (\beta^{-\alpha}).t^{\alpha}$
Gamma-based VDM [23]	$\Omega(t_0) = \int_{t=0}^{t_0} \frac{\gamma}{\Gamma(\alpha)\beta^{\alpha}} t^{\alpha-1} e^{-\frac{t}{\beta}} dt$
Weibull-based VDM [18]	$\Omega(t) = \gamma\{1 - e^{-\left(\frac{t}{\beta}\right)^{\alpha}}\}$
AML VDM [6]	$\Omega(t) = \frac{B}{BCe^{-ABt}+1}$
Normal-based VDM [23]	$\Omega(t) = \frac{\gamma}{1+e^{-\frac{(t-\mu)}{s}}}$
Rescorla Exponential (RE) [5]	$\Omega(t) = \gamma(1 - e^{-\lambda t})$
Rescorla Quadratic (RQ)[5]	$\Omega(t) = \frac{At^2}{2} + Bt$
Younis Folded (YF) [35]	$\Omega(t) = \frac{\gamma}{2}\{\mathrm{erf}\left(\frac{t-\tau}{\sqrt{2\sigma}}\right) + \mathrm{erf}\left(\frac{t+\tau}{\sqrt{2\sigma}}\right)\}$

3 MODELS USED

3.1 Vulnerability Discovery Models (VDMs)

Vulnerability discovery models (VDMs) can be characterized into two classes: time-based and effort-based VDMs. Time-based VDMs tally the vulnerabilities of a given software as a function of calendar time. Effort based VDMs, presented by Alhazmi et.al [6], consider changes in environmental factors over usage time of the software like, for example, the number of installations, share of installed base of the software and so forth (see [6] for more information regarding effort-based models). In this paper, we will focus on time-based models since the data sources we have utilized have information about vulnerability report dates but not installation dates. A classification of the time-based VDMs used in this research is shown in Figure 1, based on [20]. These models include the most well-known VDMs utilized in the literature.

S-shaped VDMs isolate the procedure of vulnerability disclosure into three phases as shown in Figure 2. Phase 1 corresponds to the learning phase, which begins from the presentation of the product and proceeds until the start of the period "Sustained Growth" as a result of expanding popularity of the software [23]. Amid the learning phase, the vulnerability discovery intensity function is an increasing function. Phase 2, or the linear phase, is the period when the majority of the vulnerabilities are discovered. The intensity function at this phase is steady and linear. Phase 3, or the saturation phase, is the period when the majority of the vulnerabilities have been detected [20]. The vulnerability discovery intensity function for phase 3 is diminishing. This phase is present only if most vulnerabilities have been discovered.

The five S-shaped VDMs in this paper consist of two right-skewed distributions (Gamma-based VDM, Younis Folded VDM), one flexible-skewed distribution (Weibull-based VDM), and two symmetrical distributions (Alhazmi–Malaiya Logistic (AML) model and Normal distribution-based model). These VDMs include the most frequent ones for the modeling process of vulnerability discovery [23].

Furthermore, we have included three non-S-Shaped VDMs: Rescorla Exponential (RE) model, Rescorla Quadratic (RQ) model, and NHPP Power-law model. More information regarding the Rescorla models can be found in [5]. When modeling the cumulative number of failures $\Omega(t)$ for software dependability/reliability assessments, models built upon a nonhomogeneous Poisson process (NHPP) are often selected. Allodi [31] demonstrated that discovered vulnerabilities may pursue a Power-law distribution. The model utilized in this paper was applied on vulnerability data as a VDM in [32], [33]. The main assumption of this model is that the number of discovered vulnerabilities pursues a nonhomogeneous Poisson process. Moreover, in NHPP-based software reliability growth models (SRGMs), the intensity function ($\omega(t) = dE[\Omega(t)]/dt$) is assumed as a monotonic function [34].

The equations associated with all the models above are provided in Table 1.

3.1.1 NHPP Power-law

When modeling the mean cumulative number of failures (MCF) (t) for software reliability evaluations, models derived from a nonhomogeneous Poisson process (NHPP) are often used. Allodi [31] showed that the vulnerability exploitation may follow a Power-law distribution. However, such models have several assumptions. The main one is that the number of detected vulnerabilities follows a nonhomogeneous Poisson process. In addition, if we consider a software as a repairable system, its intensity function $\omega(t) = dE[\Omega(t)]/dt$, is often, for simplicity, assumed a monotonic function of t. Therefore, in NHPP-based software reliability models (SRMs) or NHPP-based VDMs, the intensity function (the detection rate of software errors/the detection rate of vulnerabilities) is considered to be a monotonic function [34]. In this research, we use an NHPP Power-law model and compare it with other VDMs in terms of modeling capabilities. The equation associated with the Power-law model is presented in Table I. This model is continuous over time and has two parameters: *a (shape parameter), β (scale parameter).*

3.1.2 Gamma-based VDM

The Gamma-based VDM, derived from the Gamma distribution, belongs to the family of right-skewed distributions. It has a continuous intensity function with three parameters: a (shape parameter), β *(scale parameter), and* γ*, which represents the total number of vulnerabilities that would finally be discovered*. The equation associated with the Gamma-based VDM is presented in Table I. This distribution is only defined for $t > 0$. The shape and the scale parameters are always positive. It is expected that for the software with large values of *t*, right-skewed distributions provide better fits to vulnerability discovery data than other models [23] because of gradual reduction in the number of discovered vulnerabilities, which yields a tail on the right side of the relevant vulnerability discovery intensity function.

3.1.3 Weibull-based VDM

The Weibull-based VDM, derived from the Weibull distribution, belongs to the family of flexible-skewed distributions. This VDM was first introduced in 2007 [18]. Like the Gamma-based VDM, the Weibull-based VDM has a continuous intensity function with three parameters: a (shape parameter), β *(scale parameter), and* γ *which represents the total number of vulnerabilities that would finally be discovered*. This VDM can be symmetrical with zero skewness for a *values* around 3. For $a < 3$, this VDM is always right-skewed, while for $a > 3$, it is left-skewed. Like the Gamma-based VDM, this distribution is defined for $t > 0$.

3.1.4 AML VDM

Alhazmi–Malaiya Logistic (AML) model belongs to the family of distributions with symmetrical intensity (rate) functions. This model was first introduced in 2005 [6] and is based upon the idea that as an operating system gains market share, the attention it receives increases. Then, after experiencing a peak, it starts decreasing when a newer version is released. Overall, the AML model assumes the cumulative number of vulnerabilities is influenced by two factors: the share of the installed base (increasing factor) and the number of remaining undiscovered vulnerabilities (declining factor). The AML model has three parameters including a constant C. Parameters A and B are empirical constants and directly estimated from the dataset. *B stands for the total number of vulnerabilities that would finally be discovered.* This model is defined for time values t from the negative infinity to the positive infinity, and the parameters must be positive.

3.1.5 Normal-based VDM

The Normal-based VDM belongs to the family of distributions with symmetrical intensity/probability density functions. This model presents a distribution with zero skewness that has three parameters: μ is a location parameter, σ is a scale parameter and γ *is the total number of vulnerabilities that would eventually be discovered*. The Normal-based VDM has lighter tails on both sides in comparison to the logistic distribution used for the AML model. For a dataset with fewer vulnerabilities discovered at the beginning and at the end of a discovery process, the Normal VDM might be a better fit than the AML model [23].

3.1.6 Rescorla VDMs

In 2005, Rescorla proposed two VDMs to estimate the number of undiscovered vulnerabilities [4], [5]. In Rescorla Exponential (RE) model, γ *is the total number of vulnerabilities that would eventually be discovered* and, as time increases, Ω approaches γ. In the second model, as t grows, Ω grows quadratically, thus it is called the Rescorla Quadratic model.

3.1.7 Younis Folded (YF) VDM

The normal distribution is symmetric around its mean and is defined for a random variable that takes values from -inf to +inf. In some cases, a distribution is needed that has no negative values. Folded distributions are kinds of asymmetrical models obtained by folding the negative values into the positive side of the distribution. The folded distribution has been found usable in industrial practices such as measurement of flatness and straightens.

In the Younis folded model [35] vulnerability discovery starts at time $t = 0$ which corresponds to the release time of the software. In this model, t represents the calendar time, τ is a location parameter, σ is a scale parameter, and γ represents the number of vulnerabilities that will be eventually discovered. Compared to AML, the Folded VDM has shorter learning phase or missing learning phase which makes the normal distribution asymmetric. It results in a higher discovery rate at the beginning which may be especially applicable to the cases where the vulnerability discovery plot is in linear phase even at the beginning.

3.2 Neural Network

Neural network models (NNMs) comprise of an arrangement of algorithms for modeling and perceiving specific patterns. NNMs have been generally utilized for predictions of sequential data in time series, for example, month to month electricity demand of a city or stock price [36]–[38]. Unlike VDMs, NNMs can incorporate the nonlinearity that exists in noisy time series data. Moreover, NNMs are not based upon a specific model since they are data driven models. Thus, NNMs are flexible nonlinear data driven models with powerful predictive capability. Data driven models are helpful for the cases without a theoretical model to explain the data. In [39], it has been empirically proven that NNMs are suitable for capturing both linear and nonlinear behavior of time series.

In this paper, we utilize a feedforward NNM to forecast the number of detected vulnerabilities over time. Feedforward NNMs are widely-used forms of neural network [36] that accept a fixed number of inputs at any given time, and generate one output. We assume that the number of future vulnerabilities rely upon the number of vulnerabilities unveiled over the past periods (lags).

We utilize a single hidden-layer NNM for one step-ahead prediction. As indicated by [40], a single hidden-layer NNM is fit for approximating non-linear functions with discretionary accuracy. Figure 3 shows the structure of our NNM that comprises of three layers called input,

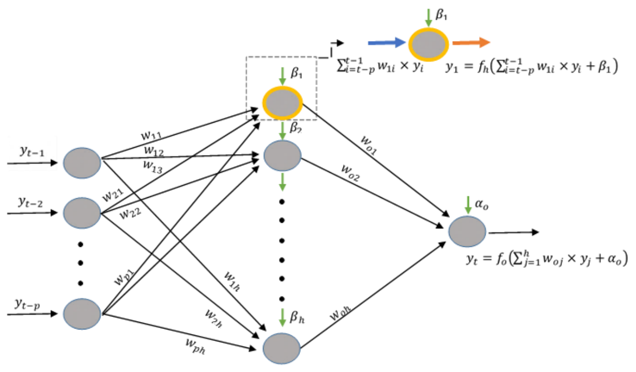


Fig. 3. The architecture of the NNM model used for our study

hidden (the middle layer), and output. Each layer is an accumulation of neurons (nodes) where the associations are governed by the corresponding weights. Data have been fed through the input layer, after that they go through the at least one hidden layer, and the ultimate result is given by the output layer.

To predict the present value, several past observations are utilized. In other words, the inputs are a p-component subset of the set $\{y_{t-p}, \ldots, y_{t-2}, y_{t-1}\}$; and y_t is the output or the total number of vulnerabilities reported in time period t. Equations 1 and 2 show the relations with the input and output values of the middle layer, consecutively. The equations associated with the output layer are represented by Equations 3 and 4, respectively.

$$I_j = \sum_{i=t-p}^{t-1} w_{ji} \times y_i + \beta_j \quad (j = 1, \ldots, h), \tag{1}$$

$$y_j = f_h(I_j) \quad (j = 1, \ldots, h), \tag{2}$$

$$I_o = \sum_{j=1}^{h} w_{oj} \times y_j + \alpha_o \quad (o = 1), \tag{3}$$

$$y_t = f_o(I_o) \quad (o = 1), \tag{4}$$

where I is the input; y denotes the output; p and h denote the number of input and hidden layer nodes, respectively; w_{ji} represents the connection weights of the input and hidden layers; and w_{oj} denotes the connection weights of the hidden and output layers. The bias values of the hidden and output layers are respectively shown by β_j and α_o, and are always between -1 and 1. f_h and f_o are the non-linear activation functions associated with the hidden and the output layers, respectively. A hyperbolic tangent function was employed as the activation function for the hidden layer since it is the function that is most widely used [36].

Deciding the optimal number of input nodes (lags) and hidden layer nodes is the initial step in structuring a NNM. From the literature, there is no systematic solution [18]. To determine the optimal number of inputs (lags) and the number of hidden nodes, we utilized the optimization algorithm (ADE-BPNN) introduced in [36]. It is shown that, for time series data, applying this algorithm improves prediction accuracy associated with basic NNMs, and other hybrid models [36]. This algorithm uses the minimum mean square error (MSE) of the training data as loss function for finding the proper number of the nodes (input and hidden) by experimentation. MSE is it the most frequently used accuracy measure in literature [41]. As our start point, we began with statistically significant lags derived from assessing the partial autocorrelation function (PACF) associated with each time series. In time series analysis, the PACF gives the linear partial correlation of a time series with its own lagged values [42]. However, we cannot only rely on the lags we found from PACF since the selection of inputs would then have be only based on the identification of a linear model, while our NNM should also be able to handle non-linear correlations. A detailed survey of existing input selection strategies for NNMs is provided in [43]. We assessed up to 50 hidden nodes for each time series and selected the number of hidden nodes that minimize the MSE.

4 Dataset Used

The dataset used in this paper was collected from the National Vulnerability Database (NVD) maintained by NIST. We leveraged the vulnerability CVE IDs to compare the reporting date of each vulnerability in NVD with the dates in other public repositories on vulnerabilities[1]. We updated the reporting dates to the earliest date that a given vulnerability was publically known in any of the vulnerability databases used [33]. To obtain exploited vulnerability data, we used Exploit Database (EDB)[2]. The EDB is a CVE compliant archive of public exploits and corresponding vulnerable software, developed for use by penetration testers and vulnerability researchers [44].

We will analyze the reported vulnerabilities associated with four well-known OSs: Windows (1995-2017), Mac (1997-2017), IOS (the OS associated with Cisco) (1992-2017), and Linux (1994-2017), as well as four well-known web browsers including Internet Explorer (1997-2017), Safari (2003-2017), Firefox (2003-2017), and Chrome (2008-2017). These software have been selected because they are the most widely used and have the most vulnerabilities among the databases. The variable we used in this research is the cumulative number of vulnerabilities reported in 30-day time intervals. In other words, for a given software, we partitioned the relative study period into intervals of 30 days, and counted the total number of vulnerabilities detected in each time interval.

For each software, all the vulnerabilities reported for any of its versions were included. For instance, all the vulnerabilities reported for mac_os, mac_os_server, mac_os_x, and mac_os_x_server were put together to create a vulnerability database for Mac.

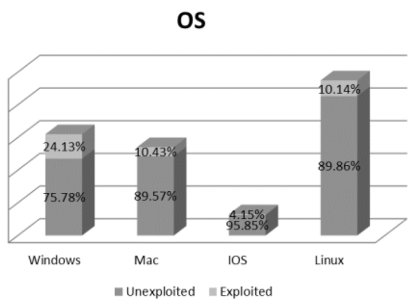
Two modeling scenarios were considered. In the first scenario (S1), we analyze all vulnerabilities reported for a software for any of its versions. In the second scenario (S2),

[1] We looked at the following ones: http://www.cvedetails.com/, https://cxsecurity.com/, http://www.security-database.com/ and http://www.securityfocus.com/

[2] https://www.exploit-db.com/

TABLE 2
NUMBER OF VULNERABILITIES PER SOFTWARE

OS	Windows	Mac	IOS	Linux
# All Vulnera-bilities	3100	2705	650	4745
# Exploited Vulnerabilities	748	282	27	481
Web Browser	**IE**	**Safari**	**Firefox**	**Chrome**
# All Vulnera-bilities	1775	943	1477	1837
# Exploited Vulnerabilities	402	108	100	78



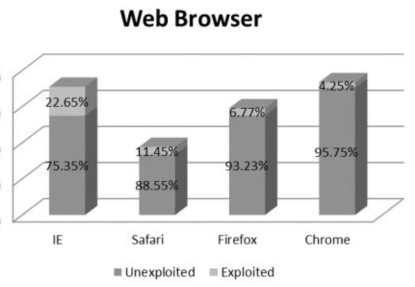


Fig. 4. Percentage of exploited vulnerabilities per software

for each software, we only consider the exploited vulnerabilities.

Table 2 presents the total number of vulnerabilities for each software (All vulnerabilities together (”S1”) and only exploited vulnerabilities (“S2”)). The percentages of exploited/unexploited vulnerabilities per software are presented in Figure 4. Windows and IE had the most percentages of exploited vulnerabilities with 24.13% and 22.65%, respectively. The reason is that, for each software, our dataset includs all the vulnerabilities reported for any of its version.

5 Analytical Steps of Scenario S1

In this section, we explain the approach we developed to predict the number of publically reported exploited vulnerabilities associated with a given software using all vulnerabilities reported for that software.

5.1 For VDMs

Regarding VDMs, we need to find a relationship between the discovery pattern of all vulnerabilities (S1) and those vulnerabilities that were exploited (S2). We focused on the ratio of the time to next vulnerability (TTNV) for exploited vulnerabilities over the TTNV associated with all vulnerabilities. TTNV was introduced as a way examining the frequency of vulnerability reports in [3]. Zhang et. al [45] also used TTNV as a measure that could imply the likelihood for presence of zero-day vulnerabilities in a software. By calculating the ratio of the TTNV for exploited vulnerabilities, which could also be referred as time to next exploit, over the TTNV associated with all vulnerabilities, we are looking for using the predictions resulted from VDMs to predict exploited vulnerabilities. In other words, we use this ratio as a multiplier in the equations associated the VDMs in the training phase to approximate the VDMs’ equations for exploited vulnerabilities. We used a resampling method and a filtering method to take care of the noisy nature of vulnerability data [46], [47]. For each software, we resample/split the vulnerability data (all vulnerabilities & exploited vulnerabilities) into intervals of 120, 150, 180, 210, 240, 270, 300, 330, 360 days to remove the effect of the daily fluctuations. For each interval (i-th interval), we calculate the mean TTNV of the observations at each time step (MTTNV) and calculate the ratio of MTTNVs, $\text{Ratio}_{interval\ i}(\text{t}) = \text{MTTNV}_{Exploited}(t)/\text{MTTNV}_{All}(t)$. Figure 5 shows the box plot of ratios associated with each interval per software. As it is shown, the median of the ratios for each software, $Median\ (\text{Ratio}_{interval\ i}(t))$, is almost constant over different intervals. The median values of the ratios per software are presented in Table 3. The VDM for exploited vulnerabilities is calculated as follows:

$$\Omega(t)_{Exploited} = \Omega(t)_{All}/Median\left(\text{Ratio}_{interval\ i}(t)\right) \qquad (5)$$

5.2 For NNM

Regarding the NNM, since we want to link two time series, we feed one time series (all vulnerabilities) into the NNM as input and select the output (y_t) from the second time series (exploited vulnerabilities). In other words, the vector of inputs $\{y_{t-p}, \dots, y_{t-2},\ y_{t-1}\}$ belongs to S1 and the output is chosen from S2.

The NNM developed for this paper was programmed utilizing Matlab R2018a. For every software, our analysis started by separating the vulnerability dataset into two groups; training and testing. The training set comprises of all the vulnerabilities published before 2015. The testing data set comprises of vulnerabilities reported in years 2015, 2016, and 2017. NNM training is a complex nonlinear optimization problem. Therefore, there is the likelihood to be caught in local minima of the error surface. To avoid getting poor outcomes, the training procedure needs to be iterated a few times with various random starting weights and biases [39].

We set the maximum training number equal to 500 epochs, which based upon our experiments provided the best results for our problem. An epoch represents the total number of times a given dataset is used for training. In other words, it represents the number of times the weights in a network were updated [48]. Since the process of model optimization in deep learning algorithms is done utilizing the gradient decent method [41], it requires to pass the training data through the network numerous times to update the weights and obtain a more precise prediction

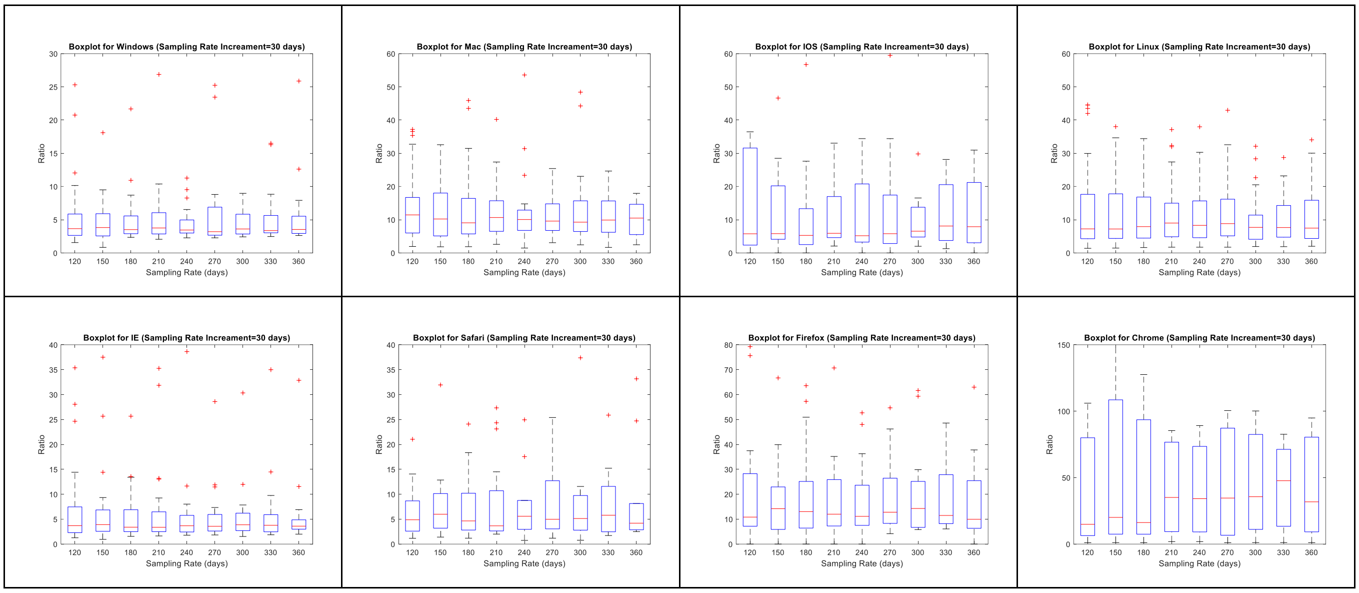


Fig. 5. Box plots of TTNV coefficient ratios per software (S2/S1)

TABLE 3
TABLE OF TTNV COEFFICIENTS (DAYS)

OS	Windows	Mac	IOS	Linux
TTNV Ratio (Median)	3.526	9.393	5.696	5.306
Web Browser	**IE**	**Safari**	**Firefox**	**Chrome**
TTNV Ratio (Median)	3.360	5.792	10.917	50.127

model [48]. As our learning function, we utilized the Levenberg-Marquardt (LM) method as recommended by the reference paper, which proposed the optimization algorithm [36]. We set the logsig and purelin functions as the activation function of the hidden and output layers, respectively. These functions also were picked similarly to the settings recommended by the algorithm. To avoid overfitting/over training, for every software, we utilized a cross validation technique by partitioning our training dataset into two subgroups of training data (70%), validation data (30%), and checked the validation performance of the trained network metrics of the Matlab Neural Network tool compartment such as gradient decent (*gradient threshold*=1.00e-4) and maximum number of validation checks (*max_fail*=100). These metrics appear as stop states of the training phase and were estimated after running a number of trials and errors while observing the training/validation error curves. Whenever the parameters of the network under training met any of these limits, the training procedure ends.

6 RESULTS

For both scenarios (S1 and S2), we used the eight VDMs for the discovery process of vulnerabilities on eight well-known software (four OSs and four web browsers). The VDMs were fitted to the datasets using a non-linear regression method described in [20]. In addition, for the first scenario (S1) we also used one NNM, which is capable of modeling nonlinearities. Since the NNM is a data driven model, we could not use it for scenario S2 due to lack of exploited vulnerabilities.

As mentioned previously, we started the analysis by splitting the data into two groups of training and test data. For scenario S1, both the VDMs and the NNM use a dataset that includes all vulnerabilities reported for all versions of a given software. For scenario S2, the VDMs use the data associated with exploited vulnerabilities reported for those versions. The training period for both scenarios starts from the time when the first exploited vulnerability associated with a given software was reported and continues until 12/31/2014. We made the predictions for the years 2015, 2016, and 2017. We then partitioned the vulnerability data into intervals of 30 days as is common in the vulnerability analysis literature [20], [21], [23].

For scenario S1, for the VDMs, during the training period, the training data was used to estimate model parameters. To avoid overfitting, 10-fold cross validation was also conducted on the training data. Using the estimated parameters and the TTNV ratios we found from Section 5, we estimated the number of exploited vulnerabilities. Then, the estimations for each time interval produced by the eight models were compared with the actual number of exploited vulnerabilities to calculate the prediction accuracy. For the NNM, for each software, we used the training data to train the NNM. The process is like feeding the NNM by one time series and comparing the outputs with values associated with another time series. Using the trained NNM, we predicted the number of exploited vulnerabilities for the next intervals. We calculated the prediction accuracy by comparing the obtained estimation and the actual number of exploited vulnerabilities.

For scenario S2, for the VDMs, during the training period, the training data was used to estimate model parameters. The estimated final values for each interval produced by the eight models were compared with the actual number of exploited vulnerabilities to calculate the prediction accuracy. The Chi-square (χ^2) goodness of fit test [20] was utilized for evaluating the quality of fit of each model on training datasets. The χ^2 statistic is calculated using the following equation:

$$\chi^2 = \sum_{i=1}^{N1} \frac{(S_i - E_i)^2}{E_i} \qquad (6)$$

where S_i and E_i are the simulated and expected observed values at i^{th} time point, respectively. N1 is the number of observations in the training dataset (the time blocks used for simulation). For acceptable fits, the corresponding χ^2 critical value should be greater than the χ^2 statistic for the given alpha level and degrees of freedom. We selected an alpha level of 0.05. The null hypothesis indicates that the actual distribution is well described by the fitted model. The p-values below 0.05 marks the fit as unsatisfactory. For each VDM, before testing its prediction capability, first we check whether the associated fit is statistically satisfactory. We neglect the model, if it shows up with a p-value<0.05.

For the training part, for the NNM, we used the MSE value to select the optimal analytical model, out of the models trained with different combination of lags. Then, for each software, the best model was selected to make the prediction for the testing dataset (the vulnerabilities reported in 2015, 2016, and 2017).

To evaluate prediction capabilities of the models, we used two common normalized predictability measures, average error (AE) and average bias (AB) [23]. AE represents how well a model predicts throughout the test phase, and AB is a measure of the model's general bias, which shows its tendency to overestimate or underestimate the number of disclosed vulnerabilities. AE and AB are defined as follows, respectively:

$$AE = \frac{1}{N2} \sum_{t=1}^{N2} \left| \frac{\Omega_t - \Omega}{\Omega} \right| \qquad (7)$$

$$AB = \frac{1}{N2} \sum_{t=1}^{N2} \frac{\Omega_t - \Omega}{\Omega} \qquad (8)$$

where *N2* presents the total number of time points (one per 30 days) over the prediction period, and Ω stands for the actual number of total exploited vulnerabilities, whereas the estimated number of total exploited vulnerabilities at interval *t* is shown by Ω_t.

For the VDMs associated with each scenario, we also report ΔVAE_i^k, which shows the difference between the AE of the i-th VDM and the VDM with minimum AE in the scenario to choose the best VDM/VDMs among the VDMs present in each scenario.

$$\%\Delta\text{VAE}_i^k = (\text{VAE}_i^k - \text{VAE}_{min}^k) * 100 \qquad (9)$$

where k is the k-th scenario, VAE_i is the AE of the i-th VDM, and VAE_{min} is the lowest AE found in the set of VDMs examined in the scenario (i.e., the best model). Thus, the ΔVAE_i^k of the best VDM in a scenario is 0.

To highlight the difference between the AE of the k-th model and the overall best model in both scenarios, we report ΔAE_i^G, which is defined as follows:

$$\%\Delta\text{AE}_j^G = (\text{AE}_j - \text{AE}_{min}^G) * 100 \qquad (10)$$

where AE_j is the AE of the j-th model, and AE_{min}^G is the lowest AE found in the set of models examined (i.e., the best model). Thus, ΔAE_j^G of the best overall model is 0. In addition, if for a given model we have $\Delta\text{AE}_j^G = 1.2$, it means than the model has 1.2% higher prediction error than the best overall model.

The Hanna and Heinold indicator (HH) is another metric to calculate prediction errors. It is proven that for some applications including analyzing real data with high fluctuation the lower values of the commonly used root mean square error (RMSE) are not always a reliable indicator of the simulations' accuracy [49]. Hence, Hanna and Heinold introduced a corrected estimator as follows [50]:

$$HH = \sqrt{\frac{\sum_{i=1}^{N2}(S_i - O_i)^2}{\sum_{i=1}^{N} S_i O_i}} \qquad (11)$$

where S_i is the i^{th} simulated data, O_i is the i^{th} observation (test data) and N represents the number of observations in test dataset (the time blocks used for simulation). The closer to zero HH is, the more accurate the model.

Tables 4-5 present the values of AE, AB, HH, ΔVAE_i^k, and ΔAE_j^G for the cases we analyzed per scenario per model (VDMs and NNM), respectively. Regarding p-values, we used * to show the models with $p<0.05$. AB can be positive (for overestimation) or negative (for underestimation), while AE is always positive. In each case, we first found the best VDMs per scenario by comparing their prediction accuracy and then compared the accuracy of those models with the NNM results. In other words, for each software, for the VDMs associated with each scenario, the models that had the smallest values of AE were selected as the best VDMs in terms of prediction and their AE values were accompanied by "bv" superscript, which stands for best VDM. In addition, the VDMs with $\Delta\text{VAE}_i^k < 2$ were also selected as the best predictive VDMs, which we assume, show similar prediction capability compared to the best VDM (the VDM/VDMs with $\Delta\text{VAE}_i^k = 0$). In other words, one of our assumptions in this paper is that the VDMs with less than 2% performance difference from the best predicting VDM represent similar prediction capabilities and the differences in their prediction performance are negligible. For each software, the best overall model in both scenarios is shown by "bo" superscripts attached to their associated AE values, which stands for best overall model (the model with $\Delta\text{AE}_j^G = 0$). If a VDM is the best model of a scenario and simultaneously is the best overall model, its AE value is only accompanied by "bo" superscript.

For each software, the normalized error values ($(\Omega_t - \Omega)/\Omega$) over prediction time are plotted in Figure 6. As is shown, the models with fewer fluctuations lead to higher accuracy.

Based upon the results provided by Tables 4-5, in terms of prediction accuracy (AE and HH), out of eight software we analyzed, scenario S1 led to the most accurate results in seven cases. Only for Firefox, the best VDM from scenario

TABLE 4 PREDICTION ACCURACY FOR OSS PER SCENARIO

Windows	*S1*					*S2*				
	AE	AB	HH	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	AE	AB	HH	$\%\Delta AE_i^2$	$\%\Delta AE_j^G$
Gamma	0.187	0.187	0.170	8.126	12.111	0.101^{bv}	-0.097	0.148	1.703	3.539
Weibull	0.148	0.148	0.142	4.245	8.230	0.139	-0.139	0.197	5.432	7.268
AML	0.106^{bv}	0.083	0.106	0.000	3.984	0.145	-0.145	0.203	6.021	7.856
Normal	0.106^{bv}	0.083	0.106	0.000	3.984	0.145	-0.145	0.203	6.021	7.856
Power-law	0.277	0.277	0.241	17.076	21.061	0.084^{bv}	0.084	0.085	0.000	1.836
RE	0.387	0.387	0.329	28.122	32.106	0.138*	0.138	0.128	NS	NS
RQ	0.274	0.274	0.239	16.766	20.750	0.113	0.113	0.108	2.892	4.727
YF	0.122^{bv}	0.118	0.123	1.576	5.560	0.140	-0.140	0.199	5.579	7.415
NNM	0.066^{bo}	-0.024	0.111	NA	0.000	NA	NA	NA	NA	NA
Mac	*S1*					*S2*				
	AE	AB	HH	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	AE	AB	HH	$\%\Delta AE_i^2$	$\%\Delta AE_j^G$
Gamma	0.215	-0.215	0.319	14.203	14.985	0.261^{bv}	-0.261	0.395	0.000	19.599
Weibull	0.251	-0.251	0.373	17.772	18.553	0.282	-0.282	0.423	2.038	21.637
AML	0.257	-0.257	0.379	18.399	19.180	0.277^{bv}	-0.277	0.416	1.577	21.177
Normal	0.257	-0.257	0.379	18.399	19.180	0.277^{bv}	-0.277	0.416	1.577	21.177
Power-law	0.073^{bv}	-0.008	0.081	0.000	0.781	0.101*	-0.008	0.113	NS	NS
RE	0.081*	0.081	0.080	NS	NS	0.092*	0.005	0.099	NS	NS
RQ	0.077^{bv}	-0.011	0.086	0.341	1.122	0.094*	0.017	0.100	NS	NS
YF	0.248	-0.248	0.368	17.513	18.295	0.280^{bv}	-0.280	0.420	1.834	21.433
NNM	0.065^{bo}	0.026	0.073	NA	0.000	NA	NA	NA	NA	NA
IOS	*S1*					*S2*				
	AE	AB	HH	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	AE	AB	HH	$\%\Delta AE_i^2$	$\%\Delta AE_j^G$
Gamma	0.149	0.146	0.165	10.441	13.206	0.032^{bv}	0.018	0.034	0.445	1.524
Weibull	0.156	0.154	0.172	11.126	13.891	0.029^{bv}	0.001	0.032	0.182	1.260
AML	0.185	0.185	0.196	14.066	16.831	0.028^{bv}	-0.014	0.036	0.000	1.079
Normal	0.185	0.185	0.196	14.066	16.831	0.028^{bv}	-0.014	0.036	0.000	1.079
Power-law	0.156	0.154	0.172	11.153	13.918	0.240*	0.240	0.217	NS	NS
RE	0.301	0.301	0.304	25.647	28.412	0.279*	0.279	0.248	NS	NS
RQ	0.044^{bv}	-0.007	0.055	0.000	2.765	0.153*	0.153	0.143	NS	NS
YF	0.220	0.220	0.231	17.555	20.321	0.028^{bv}	-0.014	0.037	0.080	1.158
NNM	0.017^{bo}	-0.002	0.020	NA	0.000	NA	NA	NA	NA	NA
Linux	*S1*					*S2*				
	AE	AB	HH	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	AE	AB	HH	$\%\Delta AE_i^2$	$\%\Delta AE_j^G$
Gamma	0.132	0.132	0.124	8.919	11.140	0.116	-0.116	0.144	7.620	9.542
Weibull	0.140	0.140	0.131	9.732	11.954	0.128	-0.128	0.159	8.874	10.796
AML	0.043^{bv}	0.037	0.054	0.000	2.221	0.168	-0.168	0.205	12.811	14.733
Normal	0.043^{bv}	0.037	0.054	0.000	2.221	0.168	-0.168	0.205	12.811	14.733
Power-law	0.182	0.182	0.168	13.914	16.135	0.040^{bv}	0.040	0.046	0.000	1.922
RE	0.282	0.282	0.255	23.969	26.190	0.045^{bv}	0.045	0.049	0.540	2.462
RQ	0.196	0.196	0.180	15.291	17.513	0.050^{bv}	0.050	0.053	1.031	2.953
YF	0.084	0.084	0.083	4.135	6.356	0.158	-0.158	0.194	11.843	13.765
NNM	0.020^{bo}	0.019	0.031	NA	0.000	NA	NA	NA	NA	NA

S2 was more accurate than the best model of scenario S1, which is NNM. In addition, considering both scenarios, the NNM was selected as the best prediction model in seven cases. As mentioned before, the VDMs with the * superscript are the models that had a p-value less than 0.5 and will not be considered in our analysis. In Tables 4-5, we used the term "NS" for these models, which stands for Not Satisfactory.

For Windows, the best model from scenario S1, which is NNM ($\Delta AE_j^G = 0$), is 1.8% more accurate than the one from scenario S2 (the model with smallest AE in scenario S2, $\Delta AE_j^G = 1.836$). For Mac, the best model is also NNM by having 19.59% smaller average prediction error (AE) than the best model from scenario S2. For IOS, Linux, IE, Safari, and Chrome the stories are like what happened for Windows and Mac by NNM (from S1) as being the best model, which comes up with 1.1%, 1.9%, 3.5%, 3.9%, and 9.3% smaller prediction errors than the best models from scenario S2. For Firefox, the model with smallest AE ($\Delta AE_j^G = 0$) belongs to scenario S2 by having 1.6% smaller AE than the best model from scenario S1, which is NNM ($\Delta AE_j^G \approx 1.6$).

Overall, scenario S1 provides more accurate results in seven cases (out of eight cases) for the number of future exploited vulnerabilities. In the only case that the best model from scenario S2 provided most accurate predictions, the performance of the best model from scenario S1 was only 1.6% worse.

Considering only VDMs, in terms of prediction accuracy (AE and HH), out of eight software we analyzed, scenario S1 led to most accurate results in only two cases. In other words, for Mac, and IE, the best VDM from S1 had higher accuracy than the best VDM from scenario S2 by having 18.8%, and 1.6% smaller prediction errors, respectively. However, the VDMs from scenario S1 were less than 2.2% different in prediction error in three cases compared to the best VDM from scenario S2. The error differences for Windows, IOS, and Linux are 2.2%, 1.6%, and 0.3%, respectively. Only for Safari, Firefox, and Chrome this difference is high and the best VDM from scenario S2 outperformed the best VDM from scenario S1 by having 16.2%, 15.7%, and 26% smaller prediction error, respectively.Overall, comparing only VDMs, scenario S1 was able to perform better than or as well as scenario S2 (with less than 2.2%

TABLE 5 PREDICTION ACCURACY FOR WEB BROWSERS PER SCENARIO

IE			***S1***					***S2***		
	AE	***AB***	***HH***	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	***AE***	***AB***	***HH***	$\%\Delta AE_i^2$	$\%\Delta AE_j^G$
Gamma	0.121	-0.121	0.132	7.189	9.018	0.175	-0.175	0.208	11.004	14.475
Weibull	0.120	-0.120	0.132	7.138	8.967	0.182	-0.182	0.217	11.700	15.170
AML	0.188	-0.188	0.220	13.922	15.752	0.256	-0.256	0.316	19.048	22.519
Normal	0.188	-0.188	0.220	13.922	15.752	0.256	-0.256	0.316	19.048	22.519
Power-law	0.120	-0.120	0.132	7.113	8.942	0.087	-0.087	0.100	2.172	5.643
RE	0.049^{bv}	-0.049	0.054	0.000	1.829	0.065^{bv}	-0.065	0.075	0.000	3.471
RQ	0.102	-0.102	0.110	5.292	7.121	0.069^{bv}	-0.069	0.080	0.386	3.856
YF	0.126	-0.126	0.140	7.717	9.547	0.228	-0.228	0.279	16.253	19.724
NNM	0.030	0.010	0.035	NA	0.000	NA	NA	NA	NA	NA
Safari			***S1***					***S2***		
	AE	***AB***	***HH***	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	***AE***	***AB***	***HH***	$\%\Delta AE_i^2$	$\%\Delta AE_j^G$
Gamma	0.498	-0.498	0.847	20.921	41.101	0.140^{bv}	-0.076	0.331	1.386	5.308
Weibull	0.528	-0.528	0.925	23.943	44.123	0.126^{bv}	-0.106	0.353	0.000	3.923
AML	0.533	-0.533	0.937	24.492	44.672	0.131^{bv}	-0.095	0.344	0.500	4.423
Normal	0.533	-0.533	0.937	24.492	44.672	0.131^{bv}	-0.095	0.344	0.500	4.423
Power-law	0.357	-0.357	0.550	6.898	27.078	0.285	0.224	0.250	15.963	19.886
RE	0.288^{bv}	-0.288	0.428	0.000	20.181	0.265	0.193	0.240	13.924	17.847
RQ	0.351	-0.351	0.541	6.312	26.493	0.270	0.202	0.242	14.462	18.384
YF	0.527	-0.527	0.923	23.863	44.043	0.127^{bv}	-0.104	0.351	0.101	4.024
NNM	0.087	0.042	0.130	NA	0.000	NA	NA	NA	NA	NA
Firefox			***S1***					***S2***		
	AE	***AB***	***HH***	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	***AE***	***AB***	***HH***	$\%\Delta AE_i^2$	$\%\Delta AE_j^G$
Gamma	0.324	0.324	0.301	15.768	31.419	0.010^{bo}	0.005	0.013	0.000	0.000
Weibull	0.324	0.324	0.301	15.743	31.394	0.029^{bv}	-0.029	0.031	1.912	1.912
AML	0.167^{bv}	0.167	0.159	0.000	15.651	0.064	-0.064	0.067	5.344	5.344
Normal	0.167^{bv}	0.167	0.159	0.000	15.651	0.064	-0.064	0.067	5.344	5.344
Power-law	0.355	0.355	0.327	18.888	34.539	0.209*	0.209	0.202	NS	NS
RE	0.492	0.492	0.436	32.510	48.162	0.199*	0.199	0.194	NS	NS
RQ	0.393	0.393	0.356	22.610	38.261	0.170*	0.170	0.169	NS	NS
YF	0.238	0.238	0.223	7.097	22.748	0.064	-0.064	0.067	5.370	5.370
NNM	0.026	-0.024	0.032	NA	1.594	NA	NA	NA	NA	NA
Chrome			***S1***					***S2***		
	AE	***AB***	***HH***	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	***AE***	***AB***	***HH***	$\%\Delta AE_i^2$	$\%\Delta AE_j^G$
Gamma	0.531^{bv}	-0.531	1.106	0.000	35.283	0.363	-0.363	0.771	9.162	18.482
Weibull	0.557	-0.557	1.180	2.635	37.918	0.359	-0.359	0.764	8.724	18.045
AML	0.544^{bv}	-0.544	1.141	1.324	36.606	0.409	-0.409	0.859	13.722	23.043
Normal	0.544^{bv}	-0.544	1.141	1.324	36.606	0.409	-0.409	0.859	13.722	23.043
Power-law	0.210*	-0.204	0.368	NS	NS	0.285^{bv}	-0.231	0.537	1.404	10.725
RE	0.108*	-0.052	0.142	NS	NS	0.271^{bv}	-0.148	0.428	0.000	9.321
RQ	0.325*	-0.325	0.579	NS	NS	0.330	-0.328	0.714	5.831	15.152
YF	0.544^{bv}	-0.544	1.143	1.275	36.557	0.473*	-0.473	0.895	NS	NS
NNM	0.178^{bo}	-0.157	0.307	NA	0.000	NA	NA	NA	NA	NA

error difference) in five cases.

Another important factor, which plays a role in model selection is the tendency of a given model to overestimate or underestimate the results. In this research, we provided the average bias values (AB) as well as the visual fluctuation trend of normalized prediction errors (Figure 6).

Now, for each software, we compare the best overall model and the models of similar prediction power (those with $\Delta AE_j^G \leq 2$), in terms of average bias. For a given software, if there are multiple models that satisfy the mentioned condition, we consider the model with lowest AB. There are five software, which are qualified for this condition (i.e. Mac, IOS, Linux, IE, and Firefox). For Linux, IE, and Firefox, the absolute value of AB for the best overall model was smaller than the other candidates with $\Delta AE_j^G \leq 2$ by 2.1%, 3.9%, and 1.9%, respectively. This For Mac and IOS, the best overall model has higher absolute bias by 1.8%, and 0.1% difference, respectively.

7 Discussion and Limitations

In terms of prediction accuracy (AE and HH), considering the OSs and web browsers (eight cases), our presented approach led to more accurate results in seven cases. Out of those cases, the NNM provided the best model in all the cases. Comparing only VDMs, in terms of prediction accuracy, scenario S1 was able to perform better than or as well as scenario S2 (with less than 2.2% error difference) in five cases.

We believe that the NNM's better execution contrasted with VDMs originates from the capacity of the NNM in foreseeing the nonlinearity nature of the vulnerability discovery process as a time series. Moreover, a common assumption in most VDMs is the pure S-shaped curve for vulnerability discovery process or considering a discovery function with a monotonic disclosure rate with constant total number of vulnerabilities. While, in reality, the vulnerability discovery process of a given software may have several linear and saturation phases as the total number of vulnerabilities may change as the result of introducing newer software versions. Furthermore, VDMs and traditional time-series functions only utilize one set of parameters for estimation. NNMs due to having multilayer perceptron structure, having various neurons per layer, and

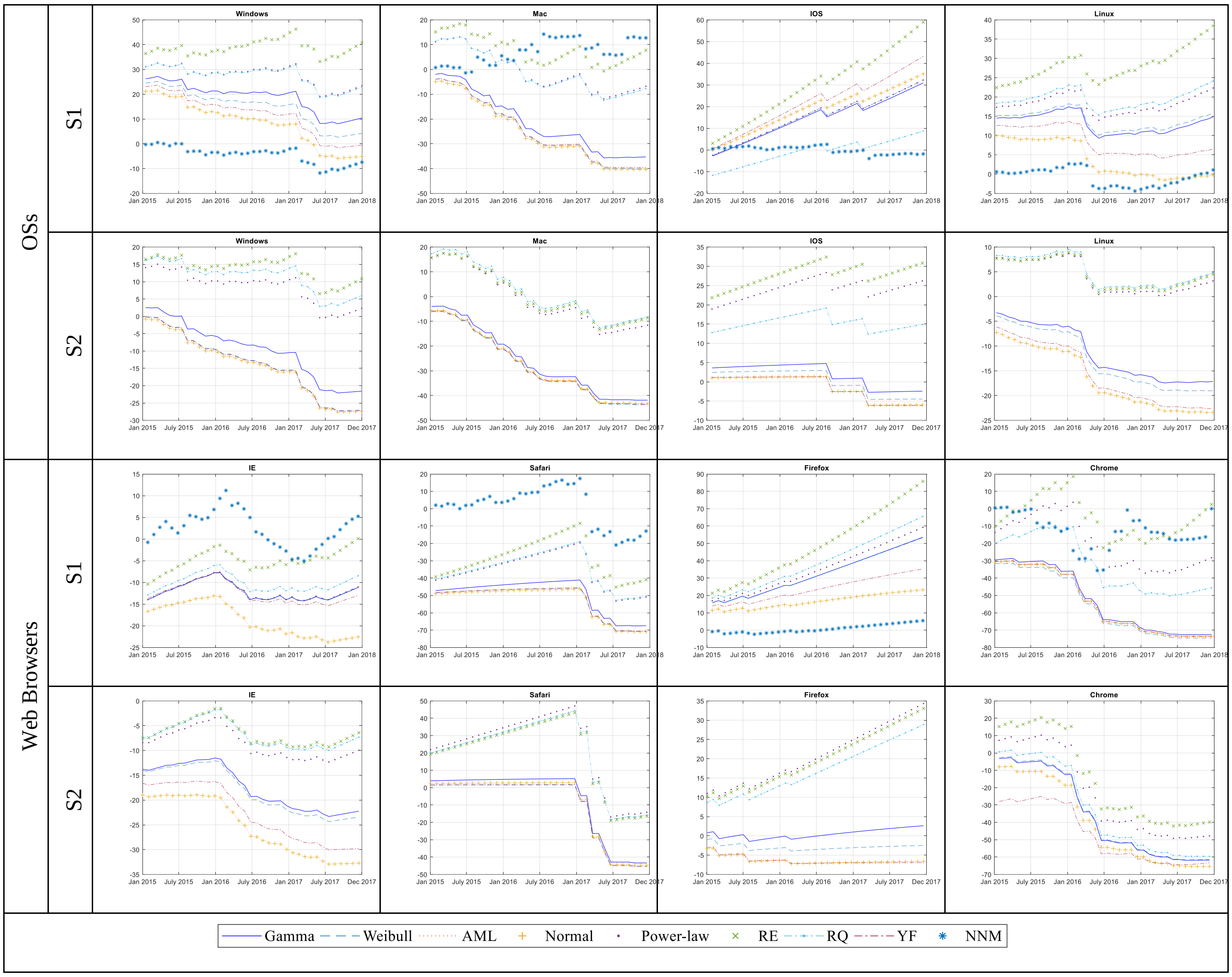


Fig. 6. Prediction errors for OSs and web browsers The X-axis indicates time (Year). The Y-axis represents normalized prediction error values in percent ($(\Omega_t - \Omega)/\Omega$).

utilizing diverse arrangement of parameters per neuron yield a structure with higher complexity for prediction.

In terms of overall magnitude of bias (i.e., absolute value of AB), out of the seven cases that scenario S1 performed better, the best model from scenario S1 outperformed the best VDMs from scenario S2 (those with $\Delta AE_j^G \leq 2$) in five cases.

We believe that, in equivalent precision conditions, in terms of bias, the final decision is up to the specialist to pick the best model dependent on his/her priorities. Nevertheless, from a security perspective, it is better to pick a model, which gives more conservative forecast results. In the current study, out of the seven NNMs that were chosen as the best models, the AB value in three cases (Windows, IOS, and Chrome) is negative. In other words, in these cases, the predictor underestimated the total number of exploited vulnerabilities. It can also be easily inferred from Figure 6, where for Windows, IOS, and Chrome most of the prediction points associated with the NNMs are located under the X=0 axis. For rest of the cases, the best overall model has come up with positive ABs or conservative results.

There are a few limitations to our work that prevents us from expanding our conclusions in a more generalized manner. One of which is with respect to utilizing announced published date of vulnerabilities as their discovery date. Vulnerabilities normally are found by pernicious users earlier than the time they are officially reported. To ensure that this gauge is as close as conceivable to the real date the vulnerability is publically known to the world, we searched for various vulnerability repositories and selected the earliest date announced for a vulnerability. Better gauges can be achieved in the event that we have more precise proxies for ascertaining attacker effort and more exact times on when a vulnerability is found and reported (for instance, in the dark web), as opposed to when it is detected in an open vulnerability database. However, acquiring this information is not straightforward: data in the dark web is unstructured and extremely hard to add significance to what is mined.

Another limitation is as to the manner in which we combined all vulnerabilities announced for all versions of a given software to have sufficient data for training the models. While a number of studies utilize vulnerability data associated with separate version of software (e.g. Windows 7) on which to apply VDMs [20], [28], there are papers that consider all versions of a software together [23], [32]. The first group expects that each version of a given software is an independent and all around characterized item, yet distinguishing the sources of reliance in vulnerability data is not a simple task.

VDMs utilize the calendar time, which may not be a decent proxy for the product utilization. In security the difficulty is in evaluating the "attacker effort" - the sum of time that a malicious user/attacker spends in finding a vulnerability - which is something that isn't required for dependability (we assume the users inadvertently experience faults that lead to failure, henceforth use time is a sufficient proxy for time between failures). A broader discussion of this limitation is addressed in [51].

Another limitation is with regard to the NNMs, since they are not mathematically tracable and easily interpretable, unlike analytical models (ie. VDMs). However, it is quite benefial to use their modeling capability as a guidance for improving the structure of the analytical models as some vulnerability discovery mechanisms might be missed by the common VDMs [52]. In this research, we showed that more accurate predictions are also possible using NNMs.

Another limitation is with regard to the availability of public information for exploits. Many vendors and public repositories, with good reason, may not publish information on exploits as that is likely to increase the security risks for the end users of those systems. Responsible hackers are also more likely to not publish their exploits in public fora, as they can report them to the vendors directly. Malicious hackers are more likely to attempt to monetize their discoveries via dark web fora. Hence the predictions we make of publically known exploits are likely to be underestimates of the true number of all vulnerabilities with exploits. Nevertheless, the approach we describe in this paper can be used by vendors and organization who have more information about exploits that they cannot share publically to calibrate their predictions.

8. Conclusion and Future Work

In this paper, we evaluated the capability of all vulnerabilities associated with a software in predicting the number of exploited ones. We compared two scenarios: S1 (use of all vulnerabilities) and S2 (use only of exploited vulnerabilities). We used eight common vulnerability discovery models (VDMs) for both scenarios as well as a non-linear neural network model (NNM) for the first scenario. Due to insufficient number of exploited vulnerabilities, it was not conceivable to use NNM for the second scenario. We used the aforementioned models for predicting the total number of future exploited vulnerabilities over a prediction period of three years. The mentioned models were applied to vulnerability data associated with four well known OSs and four well-known web browsers. We evaluated the models in terms of prediction accuracy and prediction bias. The main highlights from the results are:

- Out of eight software we analyzed, the first scenario led to more accurate results in seven cases. Moreover, out of these seven cases, the NNM was chosen as the best model in all the cases.
- Comparing only VDMs, in terms of prediction accuracy, the first scenario was able to acceptably approximate the results from the second scenario in five cases (by performing better in two cases and providing less than 2.2% error difference in three cases). This is good since we do not always have access to exploited vulnerability data, which are scarce, and need to predict their report time based on other publically accessible information.
- This study shows that neural networks are promising for accurate predictions of the number of software vulnerabilities.

For future work, we are planning on publishing the results associated with different settings we tried for our neural networks as well as other possible configurations to investigate the best neural network structure for our problem. In addition, we intend to explore other nonlinear model structures using machine learning algorithms. Among them are Recurrent Neural Network (RNN) models, used for prediction time series, which may better than NNMs at modeling dependencies between two points in a sequence. We also plan to find the reason behind the observed gap between prediction capabilities of the NNMs versus VDMs and to investigate whether current VDMs missing a mechanism associated with the process of vulnerability discovery within their mathematical structure.

9 Acknowledgements

This research is supported by NSF Award #1223634, and the UK EPSRC project D3S (Diversity and defence in depth for security: a probabilistic approach) and the European Commission through the H2020 programme under Grant Agreement 700692 (DiSIEM).

References

[1] H. Okamura, M. Tokuzane, and T. Dohi, “Optimal Security Patch Release Timing under Non-homogeneous Vulnerability-Discovery Processes,” Nov. 2009, pp. 120–128, doi: 10.1109/ISSRE.2009.19.

[2] M. R. Lyu, Ed., *Handbook of software reliability engineering*. Los Alamitos, Calif. : New York: IEEE Computer Society Press ; McGraw Hill, 1996.

[3] J. A. Ozment, “Vulnerability discovery & software security,” University of Cambridge, 2007.

[4] E. Rescorla, “Security holes... Who cares?,” presented at the USENIX Security, Aug. 2003.

[5] E. Rescorla, “Is finding security holes a good idea?,” *IEEE Secur. Priv. Mag.*, vol. 3, no. 1, pp. 14–19, Jan. 2005, doi: 10.1109/MSP.2005.17.

[6] O. H. Alhazmi and Y. K. Malaiya, “Quantitative vulnerability assessment of systems software,” 2005, pp. 615–620, doi: 10.1109/RAMS.2005.1408432.

[7] L. Allodi and F. Massacci, “A Preliminary Analysis of Vulnerability Scores for Attacks in Wild: The Ekits and Sym Datasets,” in *Proceedings of the 2012 ACM Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, New York, NY, USA, 2012, pp. 17–24, doi: 10.1145/2382416.2382427.

[8] A. A. Younis and Y. K. Malaiya, “Comparing and

Evaluating CVSS Base Metrics and Microsoft Rating System," in *2015 IEEE International Conference on Software Quality, Reliability and Security*, Aug. 2015, pp. 252–261, doi: 10.1109/QRS.2015.44.

[9] R. Adhikari and R. K. Agrawal, "An introductory study on time series modeling and forecasting," *ArXiv Prepr. ArXiv13026613*, 2013.

[10] P. E. Verissimo *et al.*, "Intrusion-tolerant middleware: the road to automatic security," *IEEE Secur. Priv. Mag.*, vol. 4, no. 4, pp. 54–62, Jul. 2006, doi: 10.1109/MSP.2006.95.

[11] H. Okamura, M. Tokuzane, and T. Dohi, "Quantitative Security Evaluation for Software System from Vulnerability Database," *J. Softw. Eng. Appl.*, vol. 06, no. 04, p. 15, Apr. 2013, doi: 10.4236/jsea.2013.64A003.

[12] W. A. Arbaugh, W. L. Fithen, and J. McHugh, "Windows of vulnerability: a case study analysis," *Computer*, vol. 33, no. 12, pp. 52–59, Dec. 2000, doi: 10.1109/2.889093.

[13] S. Frei, M. May, U. Fiedler, and B. Plattner, "Large-scale Vulnerability Analysis," in *Proceedings of the 2006 SIGCOMM Workshop on Large-scale Attack Defense*, New York, NY, USA, 2006, pp. 131–138, doi: 10.1145/1162666.1162671.

[14] S. Frei, D. Schatzmann, B. Plattner, and B. Trammell, "Modeling the Security Ecosystem - The Dynamics of (In)Security," in *Economics of Information Security and Privacy*, Springer, Boston, MA, 2010, pp. 79–106.

[15] G. R. Hudson, "Program errors as a birth-and-death process," System Development Corp., Report SP-3011, Dec. 1967.

[16] R. Anderson, "Security in open versus closed systems—the dance of Boltzmann, Coase and Moore," Cambridge University, England, Technical report, 2002.

[17] O. H. Alhazmi and Y. K. Malaiya, "Modeling the vulnerability discovery process," in *16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05)*, Nov. 2005, pp. 10 pp. – 138, doi: 10.1109/ISSRE.2005.30.

[18] J. Kim, Y. K. Malaiya, and I. Ray, "Vulnerability Discovery in Multi-Version Software Systems," in *10th IEEE High Assurance Systems Engineering Symposium, 2007. HASE '07*, Nov. 2007, pp. 141–148, doi: 10.1109/HASE.2007.55.

[19] P. L. Li, M. Shaw, J. Herbsleb, B. Ray, and P. Santhanam, "Empirical Evaluation of Defect Projection Models for Widely-deployed Production Software Systems," in *Proceedings of the 12th ACM SIGSOFT Twelfth International Symposium on Foundations of Software Engineering*, New York, NY, USA, 2004, pp. 263–272, doi: 10.1145/1029894.1029930.

[20] F. Massacci and V. H. Nguyen, "An Empirical Methodology to Evaluate Vulnerability Discovery Models," *IEEE Trans. Softw. Eng.*, vol. 40, no. 12, pp. 1147–1162, Dec. 2014, doi: 10.1109/TSE.2014.2354037.

[21] O. H. Alhazmi and Y. K. Malaiya, "Application of Vulnerability Discovery Models to Major Operating Systems," *IEEE Trans. Reliab.*, vol. 57, no. 1, pp. 14–22, Mar. 2008, doi: 10.1109/TR.2008.916872.

[22] S. Woo, O. Alhazmi, and Y. Malaiya, "Assessing Vulnerabilities in Apache and IIS HTTP Servers," 2006, pp. 103–110, doi: 10.1109/DASC.2006.21.

[23] H. Joh and Y. K. Malaiya, "Modeling Skewness in Vulnerability Discovery: Modeling Skewness in Vulnerability Discovery," *Qual. Reliab. Eng. Int.*, vol. 30, no. 8, pp. 1445–1459, Dec. 2014, doi: 10.1002/qre.1567.

[24] R. Johnston, S. Sarkani, T. Mazzuchi, T. Holzer, and T. Eveleigh, "Bayesian-model averaging using MCMCBayes for web-browser vulnerability discovery," *Reliab. Eng. Syst. Saf.*, vol. 183, pp. 341–359, Mar. 2019, doi: 10.1016/j.ress.2018.11.030.

[25] R. Johnston, S. Sarkani, T. Mazzuchi, T. Holzer, and T. Eveleigh, "Multivariate models using MCMCBayes for web-browser vulnerability discovery," *Reliab. Eng. Syst. Saf.*, vol. 176, pp. 52–61, Aug. 2018, doi: 10.1016/j.ress.2018.03.024.

[26] Y. Movahedi, M. Cukier, and I. Gashi, "Vulnerability Prediction Capability: A Comparison between Vulnerability Discovery Models and Neural Network Models," *Comput. Secur.*, p. 101596, Aug. 2019, doi: 10.1016/j.cose.2019.101596.

[27] A. Ozment and S. E. Schechter, "Milk or wine: does software security improve with age?," presented at the 15th USENIX Security Symposium, Jul. 2006, Accessed: Mar. 21, 2017. [Online]. Available: https://www.usenix.org/legacy/event/sec06/tech/full_papers/ozment/ozment_html/.

[28] V. H. Nguyen, S. Dashevskyi, and F. Massacci, "An automatic method for assessing the versions affected by a vulnerability," *Empir. Softw. Eng.*, vol. 21, no. 6, pp. 2268–2297, Dec. 2016, doi: 10.1007/s10664-015-9408-2.

[29] B. B. Madan, K. Goševa-Popstojanova, K. Vaidyanathan, and K. S. Trivedi, "A method for modeling and quantifying the security attributes of intrusion tolerant systems," *Perform. Eval.*, vol. 56, no. 1–4, pp. 167–186, 2004.

[30] H. K. Browne, W. A. Arbaugh, J. McHugh, and W. L. Fithen, "A trend analysis of exploitations," in *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*, Oakland, CA, USA, 2001, pp. 214–229, doi: 10.1109/SECPRI.2001.924300.

[31] L. Allodi, "The Heavy Tails of Vulnerability Exploitation," in *Engineering Secure Software and Systems*, Mar. 2015, pp. 133–148, doi: 10.1007/978-3-319-15618-7_11.

[32] Y. Movahedi, M. Cukier, A. Andongabo, and I. Gashi, "Cluster-based vulnerability assessment of operating systems and web browsers," *Computing*, Sep. 2018, doi: 10.1007/s00607-018-0663-0.

[33] Y. Movahedi, M. Cukier, A. Andongabo, and I. Gashi, "Cluster-based Vulnerability Assessment Applied to Operating Systems," presented at the 13th European Dependable Computing Conference, Geneva, Switzerland, Jun. 2017, Accessed: Nov. 29, 2017. [Online]. Available: http://edcc.dependability.org/.

[34] T. Y. Yang and L. Kuo, "Bayesian computation for the superposition of nonhomogeneous poisson processes," *Can. J. Stat.*, vol. 27, no. 3, pp. 547–556, Sep. 1999, doi: 10.2307/3316110.

[35] A. A. Younis, H. Joh, and Y. Malaiya, "Modeling Learningless Vulnerability Discovery using a Folded Distribution," in *Proceedings of the International Conference on Security and Management (SAM)*, Jan. 2011, pp. 617–623.

[36] L. Wang, Y. Zeng, and T. Chen, "Back propagation neural network with adaptive differential evolution algorithm for time series forecasting," *Expert Syst. Appl.*, vol. 42, no. 2, pp. 855–863, Feb. 2015, doi: 10.1016/j.eswa.2014.08.018.

[37] A. A. Adebiyi, A. O. Adewumi, and C. K. Ayo, "Comparison of ARIMA and Artificial Neural Networks Models for Stock Price Prediction," *J. Appl. Math.*, vol. 2014, pp. 1–7, 2014, doi: 10.1155/2014/614342.

[38] C. Bennett, R. A. Stewart, and C. D. Beal, "ANN-based residential water end-use demand forecasting model," *Expert Syst. Appl.*, vol. 40, no. 4, pp. 1014–1023, Mar. 2013, doi: 10.1016/j.eswa.2012.08.012.

[39] N. Kourentzes, D. K. Barrow, and S. F. Crone, "Neural network ensemble operators for time series forecasting," *Expert Syst. Appl.*, vol. 41, no. 9, pp. 4235–4244, Jul. 2014, doi: 10.1016/j.eswa.2013.12.011.

[40] A. Aslanargun, M. Mammadov, B. Yazici, and S. Yolacan, "Comparison of ARIMA, neural networks and hybrid models in time series: tourist arrival forecasting," *J. Stat. Comput. Simul.*, vol. 77, no. 1, pp. 29–53, Jan. 2007, doi: 10.1080/10629360600564874.

[41] G. Zhang, B. Eddy Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:," *Int. J. Forecast.*, vol. 14, no. 1, pp. 35–62, Mar. 1998, doi: 10.1016/S0169-2070(97)00044-7.

[42] D. N. Gujarati and D. C. Porter, *Basic Econometrics*. McGraw-Hill Irwin, 2009.

[43] R. May, G. Dandy, and H. Maier, "Review of input variable selection methods for artificial neural networks," in *Artificial neural networks-methodological advances and biomedical applications*, InTech, 2011.

[44] A. Andongabo and I. Gashi, "vepRisk-A Web Based Analysis Tool for Public Security Data," 2017, pp. 135–138.

[45] S. Zhang, D. Caragea, and X. Ou, "An Empirical Study on Using the National Vulnerability Database to Predict Software Vulnerabilities," in *Database and Expert Systems Applications*, vol. 6860, A. Hameurlain, S. W. Liddle, K.-D. Schewe, and X. Zhou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 217–231.

[46] C. N. Babu and B. E. Reddy, "A moving-average filter based hybrid ARIMA–ANN model for forecasting time series data," *Appl. Soft Comput.*, vol. 23, pp. 27–38, Oct. 2014, doi: 10.1016/j.asoc.2014.05.028.

[47] A. Phinyomark, A. Nuidod, P. Phukpattaranont, and C. Limsakul, "Feature extraction and reduction of wavelet transform coefficients for EMG pattern classification," *Elektron. Ir Elektrotechnika*, vol. 122, no. 6, pp. 27–32, 2012.

[48] S. Siami-Namini and A. S. Namin, "Forecasting Economics and Financial Time Series: ARIMA vs. LSTM," *ArXiv Prepr. ArXiv180306386*, 2018.

[49] L. Mentaschi, G. Besio, F. Cassola, and A. Mazzino, "Problems in RMSE-based wave model validations," *Ocean Model.*, vol. 72, pp. 53–58, Dec. 2013, doi: 10.1016/j.ocemod.2013.08.003.

[50] S. R. Hanna, D. W. Heinold, A. P. I. H. and E. A. Dept, and E. R. & T. Inc, *Development and application of a simple method for evaluating air quality models*. American Petroleum Institute, 1985.

[51] B. Littlewood *et al.*, "Towards Operational Measures of Computer Security," *J. Comput. Secur.*, vol. 2, no. 2–3, pp. 211–229, Jan. 1993, doi: 10.3233/JCS-1993-22-308.

[52] R. Iten, T. Metger, H. Wilming, L. Del Rio, and R. Renner, "Discovering physical concepts with neural networks," *ArXiv Prepr. ArXiv180710300*, 2018.

	Yazdan Movahedi received his PhD degree in Reliability Engineering at the University of Maryland, College Park in 2019. He also holds a Master's degree in Industrial Engineering from Isfahan University of Technology, Iran, in 2015. His research interests are in statistics and machine learning and their application in evaluating reliability of software/hardware.
	Michel Cukier is an associate professor of reliability engineering with a joint appointment in the Department of Mechanical Engineering at the University of Maryland, College Park. He is also the director for the Advanced Cybersecurity Experience for Students (ACES). His research covers dependability and security issues. His latest research focuses on the empirical quantification of cyber security. Dr. Cukier has published more than 70 papers in journals and refereed conference proceedings in those areas.
	Ilir Gashi holds PhD (2007) and BEng (Honours) (2003) degrees in Software Reliability and Computing respectively from City, University of London. He joined the Centre for Software Reliability (CSR) in July 2003, where he is currently a Senior Lecturer. He is currently (2018), a Principal Investigator in two projects which investigate the potential benefits of diversity and defence in depth for security. His research focus is on quantitative assessment of the dependability and security of software-based systems.