The Journal of Systems & Software 195 (2023) 111525

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Analysis of vulnerability fixing process in the presence of incorrect patches[☆]

Misbah Anjum[a], Shakshi Singhal[b], P.K. Kapur[c], Sunil Kumar Khatri[d], Saurabh Panwar[e,*]

[a] Amity Institute of Information Technology, Amity University, Noida, Uttar Pradesh, India
[b] Fortune Institute of International Business, New Delhi, India
[c] Amity Center for Interdisciplinary Research, Amity University, Noida, Uttar Pradesh, India
[d] Amity University Tashkent, Tashkent, Uzbekistan
[e] Department of Operational Research, University of Delhi, Delhi, India

ARTICLE INFO

Article history:
Received 15 August 2022
Accepted 29 September 2022
Available online 5 October 2022

Keywords:
Software vulnerabilities
Security bugs
Patch correctness
Intrusion detection
Patch management

ABSTRACT

Software vulnerabilities or security breaches can have consequences like leakage of sensitive information and malware execution, which are critical to network security. Consequently, eliminating security loopholes and vulnerabilities is imperative for the system administrator to counteract security attacks. Software should be thoroughly reviewed before it is released to uncover these security invasions. However, it is not feasible to identify and overcome all software failures during software testing due to external instances of software development, implementation costs, execution time, and unanticipated modifications to the specification. Security patching is a viable solution for such software systems to prevent attackers from exploiting existing vulnerabilities. Even after patch distribution and installation, it is crucial to determine whether the patch has effectively eliminated the vulnerability. Incorrect patches may lead to new security bugs, which may be malicious and disastrous for developing businesses and users. The present research aims to model the trend of patched vulnerabilities methodically by incorporating the generation of new vulnerabilities due to unsuccessful updations and encompassed bug fixes. The proposed analytical model is validated on the vulnerability databases obtained from the Common Vulnerabilities and Exposures repository. The empirical analysis yields that the present research has better forecasting efficacy than the benchmark studies.

1. Introduction

The software systems constantly evolve throughout their lifecycle (Kapur et al., 1999). During the development phases of the software lifecycle, manufacturers must implement security measures to protect it from unwanted bugs and cyber-attacks. Most of the lifecycle cost is utilized to detect and remove errors and on functionality enhancements by launching new versions of operating software systems (Paulk, 2002; Gazzola et al., 2017). These versions are revised with the release and implementation of new patches containing many tasks for maintenance (Raja and Tretter, 2011). The patching of computer software has become pervasive in recent years as the operating system and application vendors attempt to keep pace in the marketplace (Beattie et al., 2002). A patch is a set of alterations to a computer program or supporting data developed to update, fix, or improve it (Arora et al., 2006). Software patches fix existing vulnerabilities or bugs as they are found, enhancing the software's security, functionality, stability, and robustness (Hosmer, 2004; Panwar et al., 2021). According to Common Vulnerability Exposure (2014), "Vulnerability is a flaw in a software, firmware, hardware, or service component resulting from a weakness that can be exploited, causing a negative impact to the confidentiality, integrity, or availability of an impacted component or components". It will allow an attacker to execute commands as another user, pose as another entity or get access to classified information. Software vulnerabilities are triggered with malicious intent through an input tool called an *exploit* (Brumley et al., 2006). Through security flaws hackers exploit software vulnerabilities for illegitimate access to the software product (Alvi and Zulkernine, 2021). Through patching, software developers can rectify vulnerabilities and update the software at the user end without recompiling or reassembling.

Unpatched vulnerabilities are one of the most potent attack vectors to the critical infrastructure, as attackers deliberately target publicly-disclosed vulnerabilities (McDaniel and McLaughlin, 2009). According to Cybersecurity and Infrastructure Security Agency (CISA) report, in 2021, malicious cyber actors targeted internet-facing systems, such as email servers and virtual private

☆ Editor: W. Eric Wong.
* Corresponding author.
E-mail address: saurabhpanwar89@yahoo.com (S. Panwar).

https://doi.org/10.1016/j.jss.2022.111525
0164-1212/© 2022 Elsevier Inc. All rights reserved.

network (VPN) servers, with exploits of newly disclosed vulnerabilities. The increasing activity from computer attackers forces vendors to face the difficult challenge of keeping their systems up to date and secure through the distribution of software patches (Dunagan et al., 2004). Ideally, the security patch should be circulated whenever one discovers a vulnerability of the software product (Okamura et al., 2009). However, the vendor's development and distribution of security patches incur expenses. Time pressures can also result in poorly written patches that can degrade system performance and require another patch to remediate the problem (Kansal et al., 2019). For instance, Microsoft has admittedly issued security patches that have been recalled because they have caused systems to crash or were too large for a computer's capacity (Hosmer, 2004).

Following patch development, security or IT administrators must decide when to issue a patch. Installing patches as soon as they are released is not always the best choice in some enterprises. In some circumstances, suppliers may gather bug fixes for monthly or quarterly releases (Mack, 2013). If a patch is issued or deployed too soon, it increases the number of significant flaws. However, if a patch is released too late, vendors and developers will suffer financial and reputational losses and lost market possibilities. The situation worsens when the installed patch cannot adequately address the vulnerabilities. This condition is known as bad patching. Sometimes, the threat of disclosure pressurizes vendors to release the patches without proper testing that may fail after installation (Arora et al., 2008; Johnson et al., 2016). As software patches are meant to improve the performance of a software component, they come with risks and can have unintended effects. In other words, patches might introduce additional vulnerabilities while fixing one security vulnerability (McGraw, 1997).

With the automation of the debugging activities, software patches can be generated automatically to fix bugs without human intervention (Monperrus, 2018; Goues et al., 2019). However, the pressing issue that automated program repair (APR) systems struggle to address is how to verify that the generated patch is correct, given the incompleteness of available test suites (Tian et al., 2022). According to test-based generate-and-validate (G&V) APR techniques, it is declared *plausible* if a patch passes all the test cases in the test suite (Ghanbari and Marcus, 2022). Nevertheless, not all plausible patch is correct (or genuine). If a plausible patch complies with the program specification, it is said to be correct; otherwise, the patch is incorrect. Thus, the automated patch generation could lead to ineffective or damaging patches (Perkins et al., 2009). From the user perspective, applying a patch involves a tedious task and a risk that the patch causes an error like misconfiguration (Arora et al., 2006). Regardless of how quickly these problems are discovered and fixed, ineffective patches negatively influence the program by disclosing the location of vulnerabilities and the type of security issue being addressed. It serves as a template for attackers developing exploits for susceptible systems.

According to studies on exploit creation, attackers can utilize patches to uncover vulnerabilities and then use the results to create exploits for unpatched programs (Avery and Spafford, 2017). The difficulties are that patched code is sometimes not included in a future edition of a program. As a result, the same vulnerability reappears even though a prior problem was patched, increasing the number of vulnerabilities linearly. Also, there has been a rise in zero-day attacks that exploit unknown vulnerabilities. To keep the system secure or intact, the developer must ensure everything is in order before releasing a patch to the outside world. Because of constant advancement in information systems, software vulnerabilities severely threaten software producers. Moreover, the discovery of vulnerabilities has substantial implications for vendors and software users. When a vulnerability is discovered, the software vendors have to incur the cost of creating and circulating a patch to the users to update the vulnerable system (Gordon and Ford, 2000; Sen et al., 2020). Although users do not devise patches, they still incur prevention, detection, and correction costs for security breaches. Therefore, researchers have endeavored to quantify software security over the past decade to provide security solutions to the industries. The following two research questions guide the present research:

RQ1: How can the software vulnerability fixing process be assessed when the plausible patches are incorrect?

RQ2: How can the vulnerability generation paradigm be quantified due to ineffective or damaged patches?

Consequently, the present research addresses the questions mentioned above to understand software systems' security. The present study proposes a mathematical model to analyze the vulnerability patch trajectory over time, considering both successful and unsuccessful vulnerability patch rates. In addition, the proposed research considers new vulnerabilities introduced in the software system due to incorrect patches.

The rest of the paper is structured as follows: We review the literature regarding the vulnerability discovery models and patch releases for a software system in Section 2. Section 3 discusses the notation and assumption and the proposed mathematical model. A numerical illustration is provided in Section 4 for empirically validating the proposed model using an actual dataset and comparing the prediction efficiency with the benchmark studies. Section 5 provides the concluding remarks with the implications for future research.

2. Literature review

Software products are designed using various procedures and are based on diverse technologies (Kapur et al., 2011; Kumar et al., 2021). However, every piece of software has its benefits as well as drawbacks. The main concern associated with the software industry is the security of the software packages. With the advancements in new software technologies and increasing user requirements, large-scale and comprehensive software products are being developed. It has been observed that the larger projects have more amount of potential vulnerabilities (Gkortzis et al., 2021). Therefore, robust mechanism is required to secure software systems from malicious vulnerabilities. Few studies have been carried out to represent the security vulnerabilities mathematically in the recent past. The initial mathematical model for the vulnerability discovery process (VDM) was given by Anderson (2001), which is commonly known as Anderson's thermodynamic model. Anderson (2001) developed the thermo-dynamic vulnerability discovery model to estimate open and closed systems security. The model describes the discovery rate based on the mean time between failures (MTBF). Rescorla (2003) specified the economic effectiveness of finding and fixing the identified vulnerabilities for the company, especially when the black hat discovers them. Their model is based on the Non-homogeneous Poisson Process (NHPP) to evaluate the vulnerability discovery rate over time. The author suggested two statistical models, the Rescorla Exponential (RE) and Rescorla Linear or Quadratic (RL or RQ) model, to fit the different growth functions of vulnerability.

Alhazmi and Malaiya (2005) developed a logistic VDM (AML model), which quantitatively evaluates the vulnerabilities trend over time. The study also introduced a new metric similar to the fault density, i.e., vulnerability density. If the program has a high level of vulnerability density, then it is at extreme risk. The authors subsequently introduced a vulnerability discovery model based on effort (an AME model) that shows environment modifications instead of efforts expended. Sutton et al. (2007) suggested a way to find vulnerabilities with the help of fuzzy theory. However, the authors have not developed any mathematical models for vulnerability prediction. Besides, Kim et al. (2007a) discussed the vulnerability discovery process in multi-version software systems. Another notable contribution to the vulnerability discovery modeling was Joh et al. (2008), commonly known as Joh Weibull (JW) Model. They developed the vulnerability discovery model (VDM) using the Weibull distribution function. The model represents the asymmetric nature of the vulnerability discovery rate because of the skewness present in the probabilistic density function. Marconato et al. (2013) evaluated information systems security using quantitative measures. They proposed the lifecycle-based model to forecast risks and facilitate information to monitor the system's security level in operation. Another notable vulnerability discovery model was proposed by Joh and Malaiya (2014) for asymmetrical vulnerability datasets. They considered skewed distribution functions such as Weibull, Gamma, and Beta distribution functions to model the vulnerability discovery rate.

Later, Algarni and Malaiya (2014) scrutinized the factors that motivate the vulnerability discoverers to spend the effort on findings. As per the study, the discoverers are more attracted to bug bounty programs, which have become the main reason for their encouragement. To verify empirical VDMs, Massacci and Nguyen (2014) suggested a method that mainly focuses on two quantitative criteria that are quality and predictability analysis. Kapur et al. (2015) described a comparative study of vulnerability discovery modeling and its interdisciplinary nature. They have proposed two different VDMs and compared the prediction capability with existing models. Wang et al. (2017) proposed a mathematical model to capture and forecast the dynamics of the Industrial Control Systems (ICS) population and patching behavior. Few more studies have been presented in the past literature to quantify the vulnerability discovery process (Zhu et al., 2017; Kansal et al., 2019; Wang et al., 2019).

In addition to discovering vulnerabilities, patching also affects software security (Kansal et al., 2016). Patching makes it possible to fix and efficiently eliminate all vulnerabilities hampering the system information (Cavusoglu et al., 2007). Several research efforts have focused on economically portraying patch management. One of the studies suggested a mathematical cost model, which extrapolates the best time for patches to be released (Beattie et al., 2002). In the presence of provisions between confidentiality and credibility, Ioannidis et al. (2012) developed a mathematical cost model for installing security patches. Telang and Wattal (2007) study quantified damages caused by vulnerability disclosure among software vendors. The authors also analyzed the patch benefit and established the relationship between supplier and customer on the market. A study by Brumley et al. (2008) proposed that attackers could use patches to create exploits. The authors discussed two methods: one involves statically reverse-engineering the patched code to identify the vulnerabilities being addressed. Another way to dynamically analyze fixed code is to detect additional pathways created compared to unpatched code. These new pathways imply that a vulnerability in the unpatched software can be exploited by creating input following the patched program's new path. Cavusoglu et al. (2008) presented the model to determine the consequences of time-driven patch release and update policies.

The growing concerns about software security imply that security should be addressed throughout the software development process to give more cost-effective solutions (LeBlanc and Howard, 2002). Another security strategy is to count on devices not part of a software system, such as intrusion detection systems and firewalls. These security techniques can offer some protection but are only enabled once software development is completed (Gegick and Williams, 2005). Several invasions are based on a few identified assaults, indicating that the vulnerabilities are similar. Kansal et al. (2016) developed a vulnerability patch model by classifying vulnerabilities into direct, indirect, and unpatched. Their model forms the relationship between the success rate of patches and the number of patches. A few crucial studies in software engineering literature to model the vulnerability discovery and patch process mathematically are summarized in Table 1. Although few researchers have discussed the vulnerability patch models, no work has been done to identify vulnerabilities that arise while patching other vulnerabilities. There is no such literature available that would involve the discovery of new vulnerabilities caused during the patching process itself. Therefore, a novel approach is proposed to identify the newly introduced vulnerabilities analytically in the system during the patching process.

3. Research methodology

This section develops a new vulnerability patch model to predict the trajectory of patched vulnerabilities in the software system by incorporating the imperfect patching process. The proposed research development and the computational process can be summarized in the following steps:

Step 1: The problem of the software vulnerability fixing process in the presence of damaged patches and vulnerability generation is investigated and defined.

Step 2: Software products are selected, and their corresponding historical vulnerability database is extracted from the Common Vulnerabilities and Exposures (CVE) to predict the software vulnerability fixing pattern.

Step 3: The vulnerability generation behavior in software systems due to exploits caused by the incorrect patches is mathematically elucidated.

Step 4: The differential equation-based vulnerability patching model is developed to predict the vulnerability patching trend.

Step 5: The model fitting capability of the proposed model is validated using a nonlinear least square regression technique.

Step 6: The predictive efficacy of the analytical model over existing studies is tested using the cross-validation method.

Step 7: The data analysis results are exemplified, and the contribution of the proposed research to the theory of practice is presented.

The proposed study is based on the following assumptions:

i. Vulnerabilities exist and uniformly spread over all potentially vulnerable sites in the software system.
ii. The software product fails randomly due to vulnerabilities present in the software.
iii. The vendors eventually patch all the discovered software vulnerabilities.
iv. Software vulnerability patching process follows a Non-homogeneous Poisson Process (NHPP).
v. At any instant, some unsuccessfully patched vulnerabilities would always exist.
vi. The unsuccessful patches may generate additional vulnerabilities. Thus, the patching process is considered imperfect.
vii. Vulnerability discovery and patching is a deterministic process in continuous state space.

Table 1
Existing vulnerability discovery and patch models.

Vulnerability discovery model	Vulnerability discovery/patch rate	Mean value function
Rescorla Exponential (RE) Model	$\frac{d\Omega(t)}{dt} = A(B-\Omega)$	$\Omega(t) = B(1-e^{-At})$
Rescorla Linear/Quadratic (RL/RQ) Model	$\frac{d\Omega(t)}{dt} = Bt+k$	$\Omega(t) = \frac{Bt^2}{2}+kt$
Alhazmi Malaiya Logistic (AML) Model	$\frac{d\Omega(t)}{dt} = A\Omega(B-\Omega)$	$\Omega(t) = \frac{B}{1+BCe^{-ABt}}$
Joh Weibull (JW) Model	$\frac{d\Omega(t)}{dt} = B\left\{\frac{\alpha}{\beta}\cdot\left(\frac{t}{\beta}\right)^{\alpha-1} e^{-\left(\frac{t}{\beta}\right)^{\alpha}}\right\}$	$\Omega(t) = B\left\{1-e^{-\left(\frac{t}{\beta}\right)^{\alpha}}\right\}$ or $\Omega(t) = B\left\{1-e^{-At^{\alpha}}\right\}$
Joh and Malaiya Normal distribution based VDM	$\frac{d\Omega(t)}{dt} = \frac{Be^{-\frac{(t-\mu)}{\sigma}}}{\sigma\left(1+e^{-\frac{(t-\mu)}{\sigma}}\right)^2}$	$\Omega(t) = \frac{\gamma}{1+e^{-\frac{(t-\mu)}{\sigma}}}$
User dependent Vulnerability discovery Model (Kansal et al., 2017)	$\frac{d\Omega(t)}{dt} = \left\{\begin{array}{l}\left(x+y\cdot\frac{\Omega(t)}{B}\right)\\ \cdot(B-\Omega(t))\end{array}\right\}\cdot\{\phi\}\cdot\frac{dS_b(t)}{dt}$	$\Omega\ (t) = B\cdot\frac{\left(1+h\cdot e^{-(x+y)\cdot S_b(t)}\right)^{\phi}-\left((1+h)\cdot e^{-(x+y)\cdot\phi\cdot S_b(t)}\right)}{\left(1+h\cdot e^{-(x+y)\cdot S_b(t)}\right)^{\phi}}$
Vulnerability Patch Model (Kansal et al., 2016)	$\frac{dV(t)}{dt} = \left(A+\frac{C(1-\delta)V(t)}{M}\right)(M-V(t)) -\delta V(t)$	$V(t) = \overline{M}\left(\frac{1-e^{-(\overline{A}+\overline{C})t}}{1+\frac{\overline{C}}{\overline{A}}e^{-(\overline{A}+\overline{C})t}}\right)$ where $\overline{M} = M\frac{\Delta+\beta}{2C(1-\delta)};\ \overline{A} = \frac{\Delta-\beta}{2};\ \overline{C} = \frac{\Delta+\beta}{2}$ and $\beta = C(1-\delta)-A-\delta;\ \Delta=\sqrt{\beta^2+4C(1-\delta)A}$

3.1. Notation description

The following notations have been used in the paper for modeling purposes:

Parameter	Description
$\boldsymbol{V(t)}$	Expected number of patched vulnerabilities
$\boldsymbol{M}$	Potential number of vulnerabilities present in the software system
$\boldsymbol{h(t)}$	Successful patch rate of software vulnerabilities
$\boldsymbol{\delta}$	Unsuccessful patch rate of software vulnerabilities
$\boldsymbol{b}$	Scale parameter of logistic distribution function
$\boldsymbol{\beta}$	Learning parameter of the logistic distribution function

3.2. Model development

The software vulnerabilities patched process is considered as a counting process $(N(t),\ t \geq 0)$, specifically a Non-homogeneous Poisson Process (NHPP). A counting process $(N(t),\ t \geq 0)$ is said to be an NHPP with intensity function $(\theta(t),\ t \geq 0)$, if it satisfies the following conditions:

i. No vulnerabilities have been patched at $t = 0$, i.e., $N(0) = 0$
ii. The counting process $(N(t),\ t \geq 0)$ has independent increments, i.e., for $t_1 < t_2 < \cdots . < t_n$, the n random variables $N(t_1),\ N(t_2)-N(t_1), \ldots.., N(t_n)-N(t_{n-1})$ are independent.
iii. The probability of more than one patched vulnerability in a minimal time interval, Δt is negligible.

The distribution function for the random variable $N(t)$ is given as:

$$\Pr\{N(t)=k\} = \frac{[V(t)]^k}{k!}.e^{-V(t)},\quad k=0,1,2\ldots \tag{1}$$

$$V(t) = E\left[N(t)\right] = \int_0^t \theta(x)dx \tag{2}$$

where $\Pr\{\cdot\}$ and $E[\cdot]$ represents the probability and expectation, respectively; $V(t)$ denotes the *mean value function* and is defined as the expected number of vulnerabilities patched by time t, and $\theta(t)$ is called the intensity function, which indicates the instantaneous patched vulnerability rate at any time t.

Using the above assumption, the present section proposes a model that quantitatively measures the successful patch rate during the operational phase. The rate at which the patches have successfully removed the vulnerabilities is given by Eq. (3). The learning phenomenon is considered to model the hazard function of the patching process. Therefore, it is represented by an inflection S-shaped curve (Kapur et al., 2022):

$$h(t) = \frac{b}{1+\beta e^{-bt}} \tag{3}$$

where b represents the scale parameter and β denotes the learning parameter. The rate of change of patched vulnerabilities at any time is directly proportional to remaining vulnerabilities, i.e.

$$\theta(t) = \frac{dV(t)}{dt} = \frac{b}{1+\beta e^{-bt}}(M-V(t)) - \delta V(t) \tag{4}$$

The Eq. (4) is composed of two parts, the first part denotes the successfully patched vulnerabilities, and the second part represents the unsuccessfully patched vulnerabilities. While fixing the already identified vulnerabilities by the developer, new vulnerabilities can be introduced into the system. Even unsuccessful patches can be the reason for more vulnerabilities. Thus, vulnerabilities in the software system increase linearly over time and rate of change of patched vulnerabilities can be represented as:

$$\frac{dV(t)}{dt}=\frac{b}{1+\beta e^{-bt}}(M(1+\alpha t)-V(t))-\delta V(t) \tag{5}$$

where α is the proportion of newly introduced vulnerabilities. Eq. (5) can be further solved under initial conditions of $V(0)=0$ to arrive at the following closed-form solution:

$$V(t)=\frac{M}{1+\beta e^{-bt}}\left(\frac{b}{b+\delta}\right)\left(\left(1-e^{-(b+\delta)t}\right)\left(1-\frac{\alpha}{b+\delta}\right)+\alpha t\right) \tag{6}$$

The analytical model given by Eq. (6) can effectively estimate the successfully patched vulnerabilities and newly generated software vulnerabilities. Moreover, the proposed model may reduce into different mathematical models based on the following cases.

Case 1: When all the vulnerabilities are successfully patched, i.e., there are no unsuccessfully patched vulnerabilities and $\delta=0$, the Eq. (6) takes the following function form:

$$V(t)=M\left(\frac{b}{b+\delta}\right)\left(\left(1-e^{-bt}\right)\left(1-\frac{\alpha}{b}\right)+\alpha t\right) \tag{7}$$

Case 2: When no new vulnerabilities are generated, i.e., $\alpha=0$, then Eq. (6) behaves as follows:

$$V(t)=M\left(\frac{b}{b+\delta}\right)\left(\frac{1-e^{-(b+\delta)t}}{1-\beta e^{-bt}}\right) \tag{8}$$

Case 3: When both δ and $\alpha=0$, the Eq. (6) reduces to the following vulnerability discovery model:

$$V(t)=M\left(\frac{1-e^{-(b+\delta)t}}{1-\beta e^{-bt}}\right) \tag{9}$$

The proposed model captures the patching behavior of software devices. The model aims to predict and examine the patching of discovered vulnerabilities over time.

4. Empirical analysis: Parameter estimation and model validation

In this section, the description of the vulnerability dataset used for the analysis is discussed, and model fitting and predictive power are empirically validated.

4.1. Vulnerability dataset

The validation of a proposed model is carried out using the actual vulnerability data of commercial software products. The data sets were obtained from the US government repository Common Vulnerability Exposure (CVE CVE Terminology, 2014). The detailed description of data sets is as follows:

- *Oracle VM VirtualBox* (*DS 1*): It is a top corporation completely integrated with cloud applications and platform services. VirtualBox is free, open-source software (OSS) whose source code is available for public access. This data set consists of 66 vulnerabilities extracted from January 2011 to January 2017 by 29 software users. Vulnerabilities that have high severity scores are only considered. The refinement is to focus on the vulnerabilities that can harm the developer economically. These software vulnerabilities are Oracle VM Virtual Box version 4. x and 5. x. Oracle Virtual Box version 4.0 was initially released on 22 December 2010, and version 5 was released on 9 July 2015.
- *Google Chrome* (*DS 2*): Google Chrome is a web browser that offers web services to its clients by connecting them to the internet. Due to massive dependency on web platforms for day-to-day activities, these platforms are extensively used. As a result, the probability of a web attack on Google chrome is moderately high. Google Chrome has more than 1300 vulnerabilities in total that are published in CVE by May 2016. Overall, 47 vulnerabilities were observed that were reported by 46 users between September 2008 and May 2016. The data set has been filtered by the "DOS Overflow" vulnerability type with a severity greater than eight for the present analysis. The objective is to concentrate on only those vulnerabilities that have the potential to harm the developer economically. Therefore, vulnerabilities with a severity level greater than eight are considered.

4.2. Model fitting and goodness-of-fit analysis

The proposed model is fitted to the vulnerability dataset using a nonlinear least square (NLS) estimation procedure. The objective of NLS is to minimize the sum of squared deviation between the actual and predicted values. The statistical software SAS is utilized to carry out the data analysis. The parameter estimation results for both the data set are summarized in Table 2. All the parameter values are statistically significant.

Moreover, the results of goodness-of-fit criteria such as Sum of Squared Error (SEE), Mean Square Error (MSE), Root Mean Square Error, Coefficient of Determination, and adjusted R-square are evaluated to assess the predictive power of the proposed analytical model. The comparison criteria used in this study are described as follows:

a. Sum of squared Error (SSE): It is the sum of the square of the difference between observed, $V(t_i)$ and predicted values, $\widehat{V}(t_i)$ for the model over the fixed period.

$$SSE=\sum_{i=1}^{n}\left(V(t_i)-\widehat{V}(t_i)\right)^2 \tag{10}$$

b. Mean Square Error (MSE): It measures the mean deviation between the predicted values and actual observations.

$$MSE=\sum_{i=1}^{n}\left(V(t_i)-\widehat{V}(t_i)\right)^2/n-s \tag{11}$$

c. Root Mean Square Error (RMSE): It is the square root of the mean of the square of errors.

$$RMSE=\sqrt{\sum_{i=1}^{n}(V_i-V(t_i))^2/n-s} \tag{12}$$

d. Coefficient of Determination (R^2): It is used to calculate the fitness of the model to the actual data.

$$R^2=1-\frac{\sum_{i=1}^{n}\left(V(t_i)-\widehat{V}(t_i)\right)^2}{\sum_{i=1}^{n}\left(V(t_i)-\sum_{j=1}^{n}V(t_i)/n\right)^2} \tag{13}$$

where $V(t_i)$ denotes the actual data value for $i=1, 2...., n$, $\widehat{V}(t_i)$ represents the predicted data value for $i=1, 2...., n$; n is the sample size of the dataset, and s is the number of unknown parameters.

Table 2
Parameter estimation result.

Mathematical model	Data set	Estimated parameters				
		M	b	δ	α	β
Proposed model	Oracle VM Virtual box (DS1)	177.98	0.071699	0.154925	0.004314	58.50501
	Google Chrome (DS2)	87.878	0.142868	0.2545	0.004314	28.53539

Table 3
Performance measure results.

Mathematical models	Dataset	SSE	MSE	RMSE	R-Square	Adj. R-Square
Proposed model	Oracle VM Virtual box (DS1)	310.8	4.3161	2.0775	0.9883	0.9879
	Google Chrome (DS2)	207.3	2.3289	1.5261	0.9893	0.989
AML (Alhazmi and Malaiya, 2005)	Oracle VM Virtual box (DS1)	551	7.4456	2.7287	0.9793	0.9791
	Google Chrome (DS2)	307.3	3.3767	1.8376	0.9842	0.984
JW Weibull (Joh et al., 2008)	Oracle VM Virtual box (DS1)	399.3	5.3957	2.3229	0.985	0.9848
	Google Chrome (DS2)	413.9	4.5485	2.1327	0.9787	0.9784
VPM (Kansal et al., 2016)	Oracle VM Virtual box (DS1)	397.1	5.4391	2.3322	0.9851	0.9847
	Google Chrome (DS2)	281	3.1226	1.7671	0.9855	0.9852

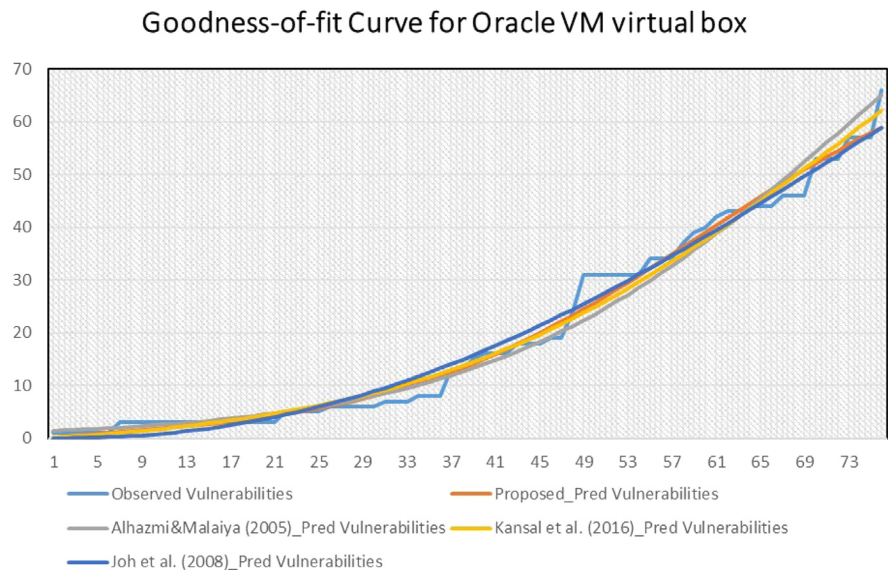

Fig. 1. Models fitting to the Oracle VM virtual box (DS1) vulnerability data.

The empirical analysis of the proposed research is also compared with three benchmark studies: Alhazmi and Malaiya (2005), Joh et al. (2008), and Kansal et al. (2016). The statistical measures for all the models are summarized in Table 3. The results listed in Table 3 exemplify that prediction errors (SSE, MSE, and RMSE) are the lowest, whereas the proposed model's R-square and adjusted R-square values are the highest. Hence, the proposed mathematical model can better predict the patched vulnerabilities for Software Projects. Besides, the goodness-of-fit curves for the models are illustrated graphically in Fig. 1 (for DS1) and Fig. 2 (for DS2). The figures demonstrate that the proposed model fits both datasets quite well.

4.3. Predictability analysis

The fitting capability of the vulnerability patch model on the dataset exhibits its practical ability. Nevertheless, it is necessary to validate the prediction efficacy of the model for future trends. The predictive power of the models is examined by conducting a cross-validation analysis. It is achieved by dividing the data into calibration (training) and forecasting (holdback or testing) periods. The first 80% of the dataset is considered for the training period, and the remaining 20% is considered for testing. The parameters are estimated by fitting the model to the training data. The parameter estimates obtained by the calibration data are then used to predict the mean value function for the testing period. The estimation method is consistent with the procedure used for model fitting purposes. The forecasted and observed values for the holdback period are compared to corroborate the model's prediction capability. Following two normalized predictability measures, average error (AE) and average bias (AB) are determined using testing data (Joh and Malaiya, 2014):

$$AE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{V(t)-\widehat{V}(t)}{V(t)}\right| \tag{14}$$

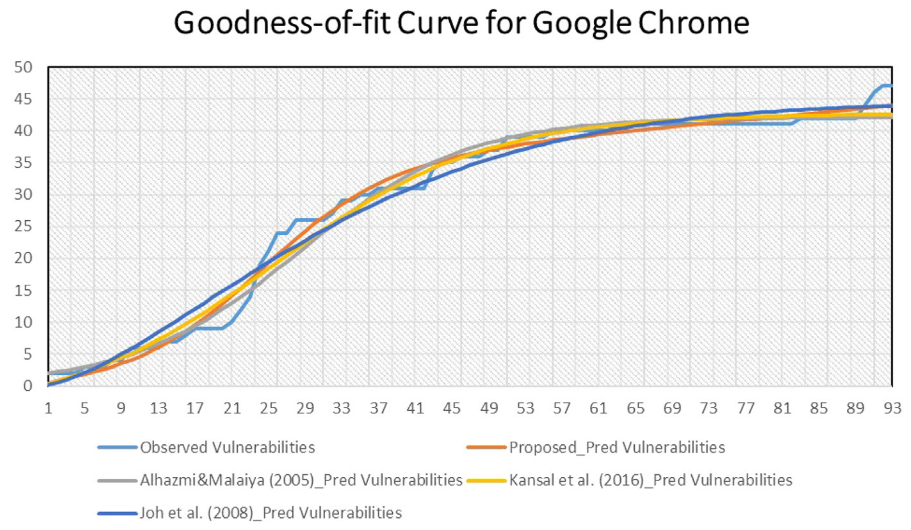

Fig. 2. Models fitting to the Google Chrome (DS2) vulnerability data.

Table 4
Prediction accuracy: Average Error and Average Bias.

Model	Dataset	Average Error (AE)	Average Bias (AB)
Proposed Model	Oracle VM Virtual box (DS1)	0.07940694	−0.07686231
	Google Chrome (DS2)	0.01930405	0.01158976
AML (Alhazmi and Malaiya, 2005)	Oracle VM Virtual box (DS1)	0.235452	−0.235452
	Google Chrome (DS2)	0.02947561	0.02462324
JW Weibull (Joh et al., 2008)	Oracle VM Virtual box (DS1)	0.08160331	0.08160331
	Google Chrome (DS2)	0.06399418	−0.05543498
VPM (Kansal et al., 2016)	Oracle VM Virtual box (DS1)	0.1049329	−0.1049329
	Google Chrome (DS2)	0.1182414	0.005488044

$$AB = \frac{1}{n}\sum_{t=1}^{n}\frac{V(t)-\widehat{V}(t)}{V(t)} \tag{15}$$

where $V(t)$ denotes the actual vulnerability and $\widehat{V}(t)$ represents the predicted number of patched vulnerabilities by time t. Average Error (AE) measures how well a model predicts throughout the testing period. Average Bias (AB) represents the general bias of the model that measures its propensity to overestimate or underestimate the number of patched vulnerabilities. Table 4 presents the results of AE and AB for all the models considered in the data analysis using testing period data. The value AE is always positive, and AB may be positive or negative. The model with the smallest AE value was selected as having the best prediction capability. From the results summarized in Table 4, it can be observed that the proposed model perform better for both the vulnerability dataset. Thus, the proposed vulnerability patch model has superior prediction capability.

5. Conclusion

The continuous up-gradation in technology has affected the security of software systems. In the recent past, the number of threats and cyber-attacks has amplified both in number and complexity. Therefore, software products require protection against threats and vulnerabilities. Although software patches are designed to improve the performance of a software component, the risk of more security breaches is associated with patching and can have unforeseen consequences. As new vulnerabilities are discovered, software patches are released to address them to avoid the risk of a security breach in the software system. However, creating and distributing a security patch costs the vendor, and time constraints can result in the release of poorly designed fixes that may decrease system performance. A practical approach is necessary to increase the number of successfully patched vulnerabilities while lowering the risks and accurately predicting the unsuccessful patching rate.

This paper proposes a new vulnerability patch model to predict the rate of successful and unsuccessful patching. The proposed model considers the imperfect patching process wherein new vulnerabilities may introduce in the system while fixing the already identified vulnerabilities by the developer. The present model measures the patched vulnerabilities quantitatively during the operational phase for security assurance and risk assessment. The proposed model is empirically validated on the actual vulnerability data of oracle and google chrome. The proposed model's performance measure result proves its prediction efficiency. Moreover, the results are compared with three benchmark studies, which exhibit the proposed study's prominence in estimation and prediction power.

This study also possesses a few limitations that open avenues for future research. In today's market dynamics, the software is released in multiple versions, with each new version having advanced features and functionality. Consequently, the proposed modeling framework can be extended to the multi-release framework. The empirical analysis can be performed in subsequent studies using a more advanced statistical technique such as the Genetic Algorithm for more reliable parameter estimation. Also, the present research can be extended to evaluate the optimal time to release a software patch, which has significant implications for software vendors and users.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Algarni, A., Malaiya, Y., 2014. Software vulnerability markets: Discoverers and buyers. Int. J. Comput. Inform. Sci. Eng. 8 (3), 71–81.

Alhazmi, O.H., Malaiya, Y.K., 2005. Modeling the vulnerability discovery process. In: 16th IEEE International Symposium on Software Reliability Engineering. ISSRE'05, IEEE, p. 10.

Alvi, A.K., Zulkernine, M., 2021. A security pattern detection framework for building more secure software. J. Syst. Softw. 171, 110838.

Anderson, R., 2001. Why information security is hard-an economic perspective. In: Seventeenth Annual Computer Security Applications Conference. IEEE, pp. 358–365.

Arora, A., Forman, C., Nandkumar, A., Telang, R., 2006. Competitive and strategic effects in the timing of patch release. In: WEIS.

Arora, A., Telang, R., Xu, H., 2008. Optimal policy for software vulnerability disclosure. Manage. Sci. 54 (4), 642–656.

Avery, J., Spafford, E.H., 2017. Ghost patches: Fake patches for fake vulnerabilities. In: IFIP International Conference on ICT Systems Security and Privacy Protection. Springer, Cham, pp. 399–412.

Beattie, S., Arnold, S., Cowan, C., Wagle, P., Wright, C., Shostack, A., 2002. Timing the application of security patches for optimal uptime. In: LISA, Vol. 2. pp. 233–242.

Brumley, D., Newsome, J., Song, D., Wang, H., Jha, S., 2006. Towards automatic generation of vulnerability-based signatures. In: 2006 IEEE Symposium on Security and Privacy. S & P'06, IEEE, p. 15.

Brumley, D., Poosankam, P., Song, D., Zheng, J., 2008. Automatic patch-based exploit generation is possible: Techniques and implications. In: 2008 IEEE Symposium on Security and Privacy. Sp 2008, IEEE, pp. 143–157.

Cavusoglu, H., Cavusoglu, H., Raghunathan, S., 2007. Efficiency of vulnerability disclosure mechanisms to disseminate vulnerability knowledge. IEEE Trans. Softw. Eng. 33 (3), 171–185.

Cavusoglu, H., Cavusoglu, H., Zhang, J., 2008. Security patch management: Share the burden or share the damage? Manage. Sci. 54 (4), 657–670.

CVE Terminology, 2014. Common vulnerabilities and exposures. Available at https://www.cve.org/ResourcesSupport/Glossary#. (Accessed date 18 June 2022).

Dunagan, J., Roussev, R., Daniels, B., Johnson, A., Verbowski, C., Wang, Y.M., 2004. Towards a self-managing software patching process using black-box persistent-state manifests. In: International Conference on Autonomic Computing, 2004. Proceedings. IEEE, pp. 106–113.

Gazzola, L., Micucci, D., Mariani, L., 2017. Automatic software repair: A survey. IEEE Trans. Softw. Eng. 45 (1), 34–67.

Gegick, M., Williams, L., 2005. Matching attack patterns to security vulnerabilities in software-intensive system designs. In: Proceedings of the 2005 Workshop on Software Engineering for Secure Systems—Building Trustworthy Applications. pp. 1–7.

Ghanbari, A., Marcus, A., 2022. Patch correctness assessment in automated program repair based on the impact of patches on production and test code. In: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. ISSTA '22, July (2022) 18–22, Virtual, South Korea, ACM, New York, NY, USA, p. 12. http://dx.doi.org/10.1145/3533767.3534368.

Gkortzis, A., Feitosa, D., Spinellis, D., 2021. Software reuse cuts both ways: An empirical analysis of its relationship with security vulnerabilities. J. Syst. Softw. 172, 110653.

Gordon, S., Ford, R., 2000. When the worlds collide: Information sharing for the security and antivirus communities. In: EICAR 2000 Best Paper Proceedings. pp. 1–20.

Goues, C.L., Pradel, M., Roychoudhury, A., 2019. Automated program repair. Commun. ACM 62 (12), 56–65.

Hosmer, C., 2004. Feasibility of Software Patch Verification. Wetstone Technologies Cortland Ny.

Ioannidis, C., Pym, D., Williams, J., 2012. Information security trade-offs and optimal patching policies. European J. Oper. Res. 216 (2), 434–444.

Joh, H., Kim, J., Malaiya, Y.K., 2008. Vulnerability discovery modeling using Weibull distribution. In: 2008 19th International Symposium on Software Reliability Engineering. ISSRE, IEEE, pp. 299–300.

Joh, H., Malaiya, Y.K., 2014. Modeling skewness in vulnerability discovery. Qual. Reliab. Eng. Int. 30 (8), 1445–1459.

Johnson, P., Gorton, D., Lagerström, R., Ekstedt, M., 2016. Time between vulnerability disclosures: A measure of software product vulnerability. Comput. Secur. 62, 278–295.

Kansal, Y., Kapur, P.K., Kumar, U., 2019. Coverage-based vulnerability discovery modeling to optimize disclosure time using multiattribute approach. Qual. Reliab. Eng. Int. 35 (1), 62–73.

Kansal, Y., Kapur, P.K., Kumar, U., Kumar, D., 2017. User-dependent vulnerability discovery model and its interdisciplinary nature. Life Cycle Reliab. Saf. Eng. 6 (1), 23–29.

Kansal, Y., Kumar, D., Kapur, P.K., 2016. Vulnerability patch modeling. Int. J. Reliab. Qual. Saf. Eng. 23 (06), 1640013.

Kapur, P.K., Kumar, S., Garg, R.B., 1999. Contributions to Hardware and Software Reliability, Vol. 3. World Scientific.

Kapur, P.K., Panwar, S., Singh, O., Kumar, V., 2022. Joint optimization of software time-to-market and testing duration using multi-attribute utility theory. Ann. Oper. Res. 312 (1), 305–332.

Kapur, P.K., Pham, H., Gupta, A., Jha, P.C., 2011. Software Reliability Assessment with OR Applications. Springer, London, p. 364.

Kapur, P.K., Yadavali, V.S., Shrivastava, A.K., 2015. A comparative study of vulnerability discovery modeling and software reliability growth modeling. In: 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management. ABLAZE, IEEE, pp. 246–251.

Kim, J., Malaiya, Y.K., Ray, I., 2007a. Vulnerability discovery in multi-version software systems. In: 10th IEEE High Assurance Systems Engineering Symposium. HASE'07, IEEE, pp. 141–148.

Kumar, V., Panwar, S., Kapur, P.K., Singh, O., 2021. Optimal decisions on software release and post-release testing: A unified approach. Yugosl. J. Oper. Res. 31 (2), 165–180.

LeBlanc, D., Howard, M., 2002. Writing Secure Code. Pearson Education.

Mack, B., 2013. Patch management overview, challenges, and recommendations, Cisco Blog. Available at: https://blogs.cisco.com/security/patchmanagement-overview-challenges-and-recommendations.

Marconato, G.V., Kaâniche, M., Nicomette, V., 2013. A vulnerability life cycle-based security modeling and evaluation approach. Comput. J. 56 (4), 422–439.

Massacci, F., Nguyen, V.H., 2014. An empirical methodology to evaluate vulnerability discovery models. IEEE Trans. Softw. Eng. 40 (12), 1147–1162.

McDaniel, P., McLaughlin, S., 2009. Security and privacy challenges in the smart grid. IEEE Secur. Priv. 7 (3), 75–77.

McGraw, G., 1997. Testing for security during development: Why we should scrap penetrate-and-patch. In: Proceedings of COMPASS'97: 12th Annual Conference on Computer Assurance. IEEE, pp. 117–119.

Monperrus, M., 2018. Automatic software repair: A bibliography. ACM Comput. Surv. 51 (1), 1–24.

Okamura, H., Tokuzane, M., Dohi, T., 2009. Optimal security patch release timing under non-homogeneous vulnerability-discovery processes. In: 2009 20th International Symposium on Software Reliability Engineering. IEEE, pp. 120–128.

Panwar, S., Kumar, V., Kapur, P.K., Singh, O., 2021. Software reliability prediction and release time management with coverage. Int. J. Qual. Reliab. Manag..

Paulk, M.C., 2002. A Software Process Bibliography. Software Engineering Institute, Last updated: October.

Perkins, J.H., Kim, S., Larsen, S., Amarasinghe, S., Bachrach, J., Carbin, M., et al., 2009. Automatically patching errors in deployed software. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. pp. 87–102.

Raja, U., Tretter, M.J., 2011. Classification of software patches: A text mining approach. J. Softw. Maintenance Evol.: Res. Practice 23 (2), 69–87.

Rescorla, E., 2003. Security holes. Who cares? In: 12th USENIX Security Symposium. USENIX Security 03.

Sen, R., Choobineh, J., Kumar, S., 2020. Determinants of software vulnerability disclosure timing. Prod. Oper. Manage. 29 (11), 2532–2552.

Sutton, M., Greene, A., Amini, P., 2007. Fuzzing: Brute Force Vulnerability Discovery. Pearson Education.

Telang, R., Wattal, S., 2007. An empirical analysis of the impact of software vulnerability announcements on firm stock price. IEEE Trans. Softw. Eng. 33 (8), 544–557.

Tian, H., Li, Y., Pian, W., Kabore, A.K., Liu, K., Habib, A., et al., 2022. Predicting patch correctness based on the similarity of failing test cases. ACM Trans. Softw. Eng. Methodol..

Wang, B., Li, X., de Aguiar, L.P., Menasche, D.S., Shafiq, Z., 2017. Characterizing and modeling patching practices of industrial control systems. Proc. ACM Measur. Anal. Comput. Syst. 1 (1), 1–23.

Wang, X., Ma, R., Li, B., Tian, D., Wang, X., 2019. E-WBM: An effort-based vulnerability discovery model. IEEE Access 7, 44276–44292.

Zhu, X., Cao, C., Zhang, J., 2017. Vulnerability severity prediction and risk metric modeling for software. Appl. Intell. 47 (3), 828–836.