Vulnerability Analysis of Sustainable Software Systems Incorporating Environmental Factor-Based Uncertainty

Divya[1], J.N.P. Singh[2], A. Anand[1*] & V.S.S. Yadavalli[3]

ARTICLE INFO

Article details
Submitted by authors 29 Sep 2025
Accepted for publication 7 May 2026
Available online 31 Aug 2026

Contact details
* Corresponding author
adarsh.anand86@gmail.com

Author affiliations
1 Department of Operational Research, University of Delhi, Delhi, India

2 Indira Gandhi National Open University, Delhi, India

3 Department of Industrial and Systems Engineering, University of Pretoria, Pretoria, South Africa

ORCID® identifiers
Divya
https://orcid.org/0009-0003-8116-5017

J.N.P. Singh
https://orcid.org/0000-0001-5395-5861

A. Anand
https://orcid.org/0000-0003-2733-3967

V.S.S. Yadavalli
https://orcid.org/0000-0002-3035-8906

DOI
http://dx.doi.org/10.7166/37-2-3310

ABSTRACT

Maintaining software's integrity and reliability throughout its lifecycle depends on effective vulnerability detection and patching, especially during the operational phase when systems face dynamic and often unpredictable environments. The complexity of real-world software patching is studied to estimate metrics such as residual vulnerabilities, detection rates, and overall system dependability. However, environmental factors, including system heterogeneity, third-party software dependencies, and variations in deployment settings, can affect the success of remedial efforts. This study presents a modelling framework that considers the influence of environmental factors on overall system performance and patching rates. The proposed approach provides a more comprehensive method for evaluating software vulnerabilities during the operational period by integrating these components. The model's descriptive and predictive capabilities are demonstrated using real-world patching data, providing valuable insights into metrics such as patching failure rates, residual vulnerabilities, and the impact of environmental uncertainty on security outcomes.

OPSOMMING

Die handhawing van sagteware se integriteit en betroubaarheid gedurende die hele lewensiklus daarvan hang af van doeltreffende kwesbaarheidsopsporing en die aanbring van regstellings, veral tydens die bedryfsfase wanneer stelsels aan dinamiese en dikwels onvoorspelbare omgewings blootgestel word. Die kompleksiteit van sagtewareregtstellings in werklike omgewings word bestudeer om maatstawwe soos oorblywende kwesbaarhede, opsporingskoerse en die algehele stelselbetroubaarheid te beraam. Omgewingsfaktore, waaronder stelselheterogeniteit, afhanklikheid van derdeparty-sagteware en variasies in ontplooiingsomgewings, kan egter die sukses van remediërende pogings beïnvloed. Hierdie studie stel 'n modelleringsraamwerk voor wat die invloed van omgewingsfaktore op die algehele stelselprestasie en regstellingskoerse in ag neem. Deur hierdie komponente te integreer, bied die voorgestelde benadering 'n meer omvattende metode vir die evaluering van sagtewarekwesbaarhede gedurende die bedryfstydperk. Die beskrywende en voorspellende vermoëns van die model word aan die hand van werklike data oor sagtewareregtstellings gedemonstreer en bied waardevolle insigte in maatstawwe soos mislukkingskoerse van regstellings, oorblywende kwesbaarhede en die invloed van omgewingsonsekerheid op sekuriteitsuitkomste.

1. INTRODUCTION

The concept of sustainability has been firmly established in the world of technology in recent years, surpassing traditional environmental concerns such as waste management, energy production, and transportation. Despite its apparent intangibility and environmental neutrality, software has a significant impact on how technology affects society and the environment. Sustainable software systems are designed to minimise environmental impact, maximise energy efficiency, and ensure long-term usability. As software systems grow more complex, they naturally become more prone to flaws that can affect their functionality, security, and reliability [1]. During their operational phase, these systems face many challenges as they adapt to constantly changing and often unpredictable conditions. External factors, such as system diversity, third-party dependencies, and different deployment environments, can significantly influence their performance and security. Detecting vulnerabilities and applying patches quickly are vital to reducing these risks, but the complexity of real-world software configurations often makes this process more difficult. As these systems become more advanced and widespread, understanding and managing their vulnerabilities becomes even more critical. Software vulnerabilities are an inherent aspect of modern technology, often resulting from external threats, coding errors, or architectural flaws. During the operational phase, they must handle a wide range of unpredictable challenges that may have an impact on their performance and security because of outside influences. Quickly identifying vulnerabilities and patching them in a timely way is essential; but the complexity of actual software environments can make these tasks quite difficult. If these flaws are not addressed, they can be exploited by malicious actors, leading to financial losses, data breaches, operational disruptions, or damage to reputation. Despite the emphasis on productivity and long-term resilience, sustainable software systems are not fundamentally immune to security threats, which is why vulnerability analysis is essential [2]. That is why this research introduces a new mathematical modelling framework that considers environmental factors and their uncertainties, aiming to improve the security and reliability of software systems throughout their operational phase.

Many organisations use patch management as a systematic and effective way to keep their systems safe and running smoothly. It involves identifying, testing, and applying updates to address security issues, which helps to protect against cyber threats such as ransomware and zero-day attacks. While patching is critical, it can be a complex process. It requires careful planning to strike a balance between maintaining system stability and addressing vulnerabilities in a timely way [3]. Rushing to patch without testing can sometimes cause new problems, such as bugs or performance issues, and the introduction of faulty patches. To do this effectively, organisations should have a well-planned patch management strategy that minimises disruptions and promptly addresses critical vulnerabilities. Regular monitoring, risk assessment, testing in controlled environments, and scheduled updates are all key parts of a practical approach. Using automation tools can help reduce errors and maintain consistency, particularly for smaller businesses. It is also crucial to follow rules and regulations to ensure compliance, which helps to protect the company's reputation and avoid legal penalties. Thus, effective patch management is a vital part of a robust security strategy, enabling organisations to stay safe and compliant in an ever-evolving digital landscape.

Handling patch management during a software's operational phase can be significantly different from handling it during development or testing [4]. The environments where live systems run are often diverse, constantly changing, and yet predictable at times. These systems might operate on different hardware setups, work with third-party software, and be influenced by user behaviours and workloads [5]. Owing to this variability, patching can present unique problems, as a patch that functions well in one situation may cause unexpected issues elsewhere. The rapid pace of patching efforts can also make it difficult for IT teams to understand fully how a patch affects the system, whether it addresses a vulnerability, has an impact on performance, or causes compatibility issues with critical third-party applications. Because of these notable differences, creating a one-size-fits-all patch management approach is not feasible. Traditional methods depend on regular patching with minimal risk, assuming that environments are standardised and controlled, which is often not the case in real life. To care effectively for the unique needs of live systems, IT teams should thoroughly test patches under conditions that closely mirror the actual working environment before deploying them widely. Complementary strategies, such as mitigation plans, rollbacks, and continuous monitoring, help to reduce potential disruptions and keep systems running smoothly.

Businesses could enhance their patching processes by adopting automated and adaptable patch management systems. These tools analyse system differences, test patches in various environments, and deploy them carefully to minimise the chance of failures. For patch management to work smoothly, it is essential to strike a balance between addressing security issues promptly and maintaining system stability without unexpected downtime. The success of these efforts often depends on environmental factors. For

example, “system heterogeneity” refers to the differences in hardware and software setups in different locations. Sometimes, a patch intended for a specific hardware or operating system may not work effectively on other configurations, resulting in failures or persistent vulnerabilities. The use of third-party software can also add layers of complexity. A patch that works perfectly in a controlled lab setting might not perform as well in the real world, where networks are less predictable and user behaviour varies [6]. Many modern applications rely on external libraries, frameworks, and application programming interfaces, which can sometimes have vulnerabilities or compatibility issues. To address these challenges effectively, it is helpful to collaborate with third-party providers or vendors, keeping in mind that their availability and timing can be uncertain. In addition, the process of deploying, configuring, and maintaining software also influences the various environments in which the system operates.

For instance, software running in the cloud encounters unique challenges compared with traditional on-premises solutions. Cloud infrastructure offers on-demand resources, and is naturally more flexible and ever-changing, which can change the environment’s state. When a software system undergoes changes between the time a vulnerability is discovered and when a patch is applied, these changes can have an impact on the effectiveness of the patch in that specific environment [7]. Even if the system environment remains relatively stable, factors such as network setup, security policies, and user interaction and behaviour can have a significant impact on how the system responds when the patch is implemented. Overall, these environmental factors create uncertainties that many traditional patch management approaches often overlook. Most existing methods focus on metrics such as the time it takes to deploy patches, vulnerability detection rates, or system uptime; however, they don’t account for environmental differences. As a result, these approaches might give an incomplete or even misleading picture of reliability and security. To fill this gap, there is a need for a new framework that considers environmental factors and the uncertainties they bring into the patch management process. Figure 1 below illustrates the software patch management process to mitigate uncertainty.

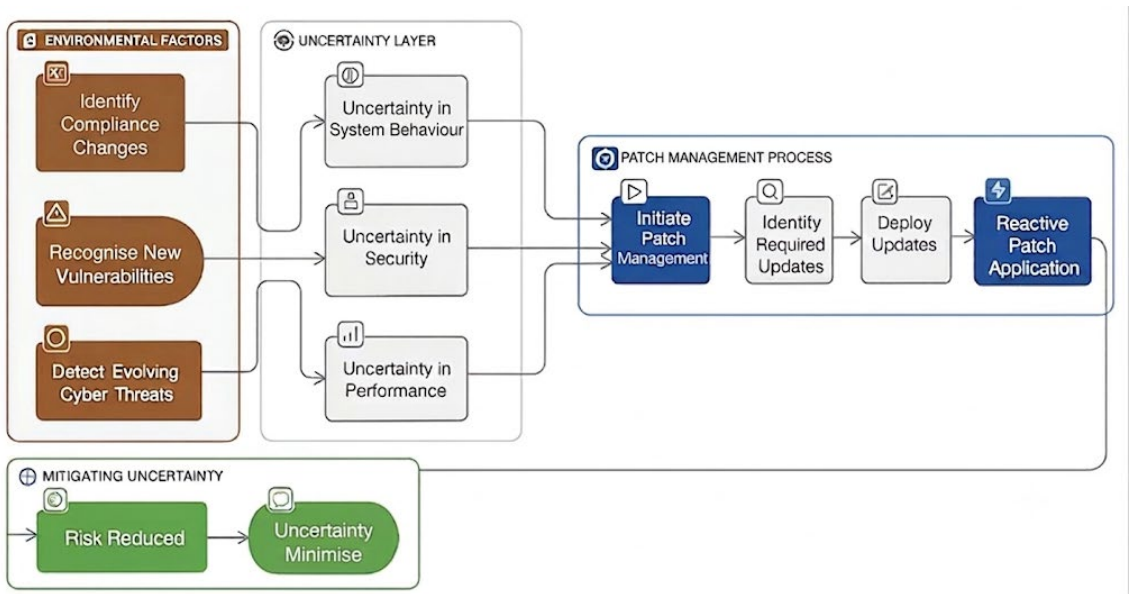

Figure 1: Software patch management process to mitigate uncertainty

This study introduces a new mathematical modelling framework that is designed to overcome the limitations by incorporating environmental factors and uncertainties. This approach is more practical and versatile for assessing software reliability and security in real-world operational settings. The proposed framework has a significant impact on software development and operations teams. The real-world case study illustrates the descriptive and predictive aspects of the proposed model, providing potential metrics related to patch failure rates, residual vulnerabilities, and the implications of environmental uncertainty for security outcomes. These insights enable organisations to optimise their patching strategies, mitigate risks, and facilitate informed decision-making in dynamic operating environments. As software systems become more complex and larger in scale, the proposed framework should help to maintain their integrity and reliability throughout their lifecycle.

Existing NHPP-based reliability models primarily focus on fault-detection processes, whereas the proposed modelling framework explicitly models patch-deployment dynamics under environmental uncertainty. Furthermore, while previous studies incorporate environmental variability into failure processes, the present study integrates:

i. Environmental uncertainty that directly affects the patch deployment rate.
ii. A learning-based dynamics patching mechanism.
iii. Validation using the real-world patch datasets rather than traditional failure datasets.

This combination provides a more realistic representation of operational phase vulnerability mitigation.

This study is divided into several sections. Part 2 reviews relevant previous work and provides a comprehensive literature survey. Section 3 introduces a modelling framework for the research. Model validation is covered in Section 4. In Section 5, the authors analyse the findings. The study concludes with Sections 6 and 7, which discuss the conclusions, future directions, and limitations.

2. LITERATURE REVIEW

Vulnerability discovery models (VDMs) are mathematical constructs that can explain and predict the discovery of security vulnerabilities in software systems. VDMs help to analyse all data used to discover vulnerabilities and also to establish a secure management plan. One of the first and most fundamental VDMs is given by Rescorla [8], who proposed that the growth of vulnerability discovery follows an exponential pattern for a specific period, then begins to decline as the number of undiscovered flaws reaches a turning point.

Building on this concept, Anand et al. [9] introduced a versatile framework for vulnerability discovery modelling that highlights vulnerabilities through three stages: learning, a linear progression, and saturation. Alhazmi's [10] further research explored how systematic quantitative approaches can improve software security analysis, while Massacci and Nguyen [11] created a method to assess the reliability and effectiveness of VDM. Kaur et al. [12] looked into delays in fixing interconnected vulnerabilities. Movahedi et al. [13] developed an innovative neural network-based approach to vulnerability detection. Liu and Xing [14] shared a 2021 framework for analysing the survivability and vulnerabilities of cloud RAID systems during disk failures and attacks. Nappa et al. [15] examined how shared code influences vulnerability identification and patching. In 2016, Anand and Bhatt [16] proposed a ranking system for VDMs based on weighted criteria, and Sharma et al. [17] compared vulnerability discovery processes between open-source and closed-source software. In 2017, Bhatt et al. [18] modelled and described software vulnerabilities, and Anand et al. [19] introduced several variants of vulnerability discovery models aimed at improving software reliability. Last, in 2017, a comprehensive empirical study on security patches was carried out by Li et al. [20].

Although a comprehensive body of research exists related to vulnerability modelling, research focusing on patches and patching has been less extensive. In 2017, Anand et al. [21] developed a software product scheduling model that highlights the critical role of patching activities in reducing outages and enhancing system performance. In 2018, Srivastava and Sharma [22] introduced another insightful vulnerability modelling scheme, which categorises vulnerabilities into direct and indirect, based on the time lag between discovery and assessment. In 2019, Almukaynizi et al. [23] proposed a more comprehensive approach for identifying the specific types of software vulnerabilities that affect a system. That same year, Anand et al. [24] also suggested a practical way to evaluate whether software systems met safety integrity level (SIL) standards by analysing post-deployment patch release schedules through a linear plot. In 2020, Anjum et al. [25] proposed an innovative framework for assessing software vulnerabilities using the best-worst method and two-way analysis.

Research on patch management models has uncovered several key factors, including the frequency of patch releases, the timing of their release, and the efficiency of their deployment. Kansal et al. [26] introduced a helpful mathematical framework that categorises vulnerabilities as directly patched, indirectly addressed, or left unresolved. Anand et al. [27] investigated how uncovering underlying vulnerabilities affects the number of patches required. Shrivastava et al. [28] shared a two-dimensional model that links patched vulnerabilities with patch release schedules. Later, Anand et al. [29] created a patch management model based on chain rule adjustments. Aggarwal et al. [30] even explored how changing vulnerabilities influence patching strategies. These studies emphasise the importance of proactive patch management and aligning the timing of patch releases with ongoing vulnerability discovery. In 2024, Divya et al. [31]

examined how to differentiate between patches and faulty patches. Anand et al. [32] also conducted a 2020 study on reliability modelling for multi-version software systems and their impact on compromised patching. Most recently, Bhatt et al. [33] investigated the selection of the best software vulnerability scanner using an intuitionistic fuzzy set approach. In 2025, Anand et al. [34] proposed a comprehensive, multi-phase framework for patch management and vulnerability detection.

Back in 2006, Teng and Pham [35] introduced an innovative approach for predicting software reliability in unpredictable environments. Then, in 2014, Chang et al. [36] developed a testing-coverage model that thoughtfully considers the uncertainties of real-world operating conditions. Similarly, in that same year, Pham [37] presented a new software reliability model featuring a V-shaped fault-detection rate, also taking into account the unpredictability of operating environments. Moving to 2016, Pham and colleagues [38] expanded on this with a generalised fault-detection model tailored to variable operating surroundings. In 2017, Li et al. [39] examined the NHPP software reliability model, addressing uncertainties caused by imperfect debugging and testing coverage. Finally, in 2018, Zhu and Pham [40] developed a model that effectively integrates a martingale process with gamma-distributed environmental factors, thereby deepening the understanding of software reliability in complex settings.

The abovementioned studies successfully integrate environmental uncertainties into software reliability modelling. They primarily focus on fault discovery rather than on the nuances of the patch deployment lifecycle. Existing patching models often rely on static assumptions about the patching rate. The proposed framework clearly differentiates itself by adapting the NHPP explicitly to incorporate a learning parameter in the patch deployment rate. This approach bridges the gap between theoretical vulnerability discovery and the practical, dynamically improving nature of operational patch deployment under environmental factors.

Table 1: Contribution of the proposed study

Study	Patch modelling	Environmental factors	Uncertainty handling	Random variable modelling	Learning effect	Focus on patch deployment
Rescorla [8]	✕	✕	✕	✕	✕	✕
Anand *et al.* [9]	✕	✕	✕	✕	✓	✕
Alhazmi [10]	✕	✕	✕	✕	✕	✕
Massacci and Nguyen [11]	✕	✕	✕	✕	✕	✕
Nappa *et al.* [15]	✓	✕	✕	✕	✕	✓
Anand *et al.* [21]	✓	✕	✕	✕	✕	✓
Kansal *et al.* [26]	✓	✕	✕	✕	✕	✓
Pham [37]	✕	✓	✓	✓	✕	✕
Li and Pham [39]	✕	✓	✓	✓	✕	✕
Zhu and Pham [40]	✕	✓	✓	✓	✕	✕
Divya *et al.* [31]	✓	✕	✕	✕	✕	✓
Proposed Study	✓	✓	✓	✓	✓	✓

Building on the work of Li and Pham [41], this new framework also offers exciting advances in software patch management. It helps us to understand how environmental factors influence patch effectiveness, considering factors such as patch failure rates, lingering vulnerabilities, and overall system trustworthiness. By incorporating these aspects, the model provides a more detailed picture of system reliability and security, enabling organisations to identify potential risks and plan more effective responses. In addition, the framework accounts for the inherent uncertainty of environmental factors, which can vary over time and in different deployments. It uses probabilistic methods to estimate the likelihood of various outcomes, such as whether a patch will succeed or fail. This approach empowers organisations to make smarter decisions, even in uncertain situations, and to prepare contingency plans for potential patch issues.

3. PROPOSED MODELLING FRAMEWORK

Effective patch management plays a vital role in protecting software systems from security threats. It is crucial during maintenance, when changing conditions can make it harder to spot and fix vulnerabilities. In everyday practice, rolling out patches can face problems such as delays, malfunctions, diverse systems, and dependencies on other components. These hurdles create uncertainties that could weaken security defences. Traditional patching methods often fall short by assuming that everything remains stable, overlooking the real-world unpredictability. To address these challenges, the authors propose a mathematical modelling approach that takes into account the natural unpredictability in patch deployment and the resolution of security issues.

3.1. Assumptions

The following presumptions form the basis of the suggested modelling framework:

1. Vulnerability detection and fixation are non-homogeneous processes that vary over time.
2. The impact of operational constraints is determined by the patch deployment rate, which is proportional to the number of unpatched vulnerabilities.
3. Assuming no delays in vulnerability detection, patches are applied as soon as they are found.
4. An environmental element, represented as a random variable, influences the patching process by taking into account external factors such as system heterogeneity, reliance on third-party software, and changes in deployment settings.

3.2. Notations

$P(t)$	the cumulative number of vulnerabilities patched by time t.
$q(t)$	patch deployment rate.
A	total number of patches to be deployed.
η	random factor affecting patch deployment, capturing uncertainty in deployment rate.
$f(\eta)$	PDF of η, representing environmental factors affecting detection.
$X(t)$	the expected number of unpatched vulnerabilities at time t.

3.3. Model development

The integration of mathematical modelling to guide the discovery process has been demonstrated in the literature on the patch deployment process. In this study, the patching process is influenced by environmental factors, and, to account for this uncertainty, the authors have proposed a modelling framework that can be expressed as equation (1). Researchers have used the non-homogeneous Poisson process (NHPP) to tackle this issue.

$$\frac{dP(t)}{dt} = \eta q(t)(A - P(t)) \tag{1}$$

Let us assume that η's probability density function (PDF) is $f(\eta)$ and η is a time-dependent variable, and it is also unit-free.

To solve equation (1), let

$$X(t) = A - P(t) \tag{2}$$

where $X(t)$ represents the unpatched vulnerabilities at time t.

Differentiating equation (2)

$$\frac{dX(t)}{dt} = -\frac{dP(t)}{dt} \tag{3}$$

Substituting from equation (1):

$$\frac{dX(t)}{dt} = -\eta q(t) X(t) \tag{4}$$

After using the initial conditions, $X(0) = A$, equation (4) becomes

$$X(t) = A\, exp(-\eta \int_0^t q(u)\, du) \tag{5}$$

The cumulative number of vulnerabilities patched by time t is represented by equation (6):

$$P(t) = A[1 - \ exp(-\eta \int_0^t q(u)\, du)] \tag{6}$$

After solving equation (6), a generalised solution can be obtained, as follows:

$$P_\eta(t) = A \int_0^t \eta q(z) e^{-\int_0^z \eta q(u) du}\, dz \tag{7}$$

Also, the expression for $P(t)$ can be obtained as

$$P(t) = \int_0^\infty P_\eta(t) f(\eta)\, d\eta \tag{8}$$

To solve equation (8), the authors used the Laplace transformation, and the closed-form solution for $P(t)$ can be given as:

$$P(t) = q[1 - F^*(S)] \tag{9}$$

Here, $F^*(S)$ is the Laplace transform, which can be given as $F^*(S) = \int_0^\infty e^{-S\eta} f(\eta)\, d\eta$.

By substituting different PDFs $f(\eta)$, the total number of required patches A, and the patch deployment rate function $q(t)$, the authors next derive different mean value functions that characterise the patching process.

The Laplace transform offers two key advantages in this framework. First, it avoids explicitly solving the integral for each probability distribution, thereby simplifying the derivation of the mean value function. Second, it provides a unified analytical structure that allows different environmental conditions to be incorporated by simply substituting the corresponding Laplace transform.

Also, the Laplace transform captures the cumulative effect of environmental variability on the patch deployment process. Because η represents uncertainty in deployment efficiency, its distribution influences the rate at which vulnerabilities are resolved. Through the Laplace transform, this stochastic variability is directly incorporated into the model, enabling the derivation of closed-form expressions for the expected number of patched vulnerabilities.

In addition, the Laplace transform preserves important properties such as monotonicity and boundedness, ensuring that the derived mean value function remains mathematically consistent with real-world patching behaviour. This quality makes it especially useful for modelling systems in which uncertainty changes over time and affects operational performance. Overall, using the Laplace transform is crucial for linking the random environmental factors to the predictable patch deployment model, thereby improving its ease of analysis and practical usefulness.

It is important to note that any non-negative distribution can be used to model the random environmental factor η, which represents uncertainties affecting patch deployment. Most existing models assume that η follows a Gamma distribution because of its flexibility in representing various operational conditions:

- If $\eta > 1$, the operational environment enables faster patch deployment than previously tested.

- If $\eta = 1$, the deployment conditions match those in a controlled testing environment.
- If $0 < \eta < 1$, deployment happens under constraints, which makes applying patches slower and less efficient than testing.

With this flexibility, the model can manage a broad spectrum of real-world patching scenarios, including automated versus manual patching, differing deployment efficiency between systems, and the effects of delayed patch rollouts.

The authors considered $f(\eta)$ as γ - distribution (Gamma distribution); then

$$f(\eta) = \frac{\lambda^k \eta^{k-1} e^{-\lambda\eta}}{k!} \tag{10}$$

Here, λ and k are the shape parameters of γ - distribution.

Using equation (5), the mean value function can be given as

$$P(t) = A\left[1 - \left(\frac{\lambda}{\lambda + \int_0^t q(u)\,du}\right)^k\right] \tag{11}$$

Furthermore, the authors discussed three cases based on the mean value function given in Equation (6).

Case 1: Constant deployment rate $q(\mu) = \mu$

The authors considered $q(\mu) = \mu$, and q remains constant in the proposed modelling framework given in equation (6); then

$$P(t) = A\left[1 - \left(\frac{\lambda}{\lambda + qt}\right)^k\right] \tag{12}$$

Here, b represents a constant patch deployment factor, meaning that the patching rate stays steady regardless of external conditions. The function suggests that patch deployment typically follows a predictable pattern, with the number of patched vulnerabilities increasing steadily over time. This scenario is more likely in systems where patch automation is well-controlled and where external factors, such as network delays, compatibility issues, or administrative restrictions, don't hinder the deployment process.

Case 2: Deployment rate influenced by quadratic growth $q(\mu) = \frac{q^2 t}{1+qt}$

In this case, the patch deployment rate is given as $q(\mu) = \frac{q^2 t}{1+qt}$, which introduces a quadratic factor in patch deployment. The corresponding mean value function can be given as

$$P(t) = A\left[1 - \left(\frac{\lambda}{\lambda + (qt - log(1+qt))}\right)^k\right] \tag{13}$$

This equation describes scenarios in which patch deployment begins slowly but accelerates as more vulnerabilities are identified and addressed. The quadratic dependency here suggests that the initial patching phase might be slow because of testing and validation, but patching efficiency improves once the process is streamlined. This applies to large-scale enterprise patching, where early rollouts involve extensive testing, but later patches can be deployed more quickly as organisations become more confident in patch reliability.

Case 3: Deployment rate with learning parameter $q(\mu) = \frac{r}{1+\beta e^{-qt}}$

In this case, the patch deployment rate is modelled as $q(\mu) = \frac{r}{1+\beta e^{-qt}}$. This equation introduces a learning parameter, β, that influences the evolution of the patching process. The corresponding mean value function is:

$$P(t) = A\left[1 - \left(\frac{\lambda}{\lambda + log(\frac{e^{qt}+\beta}{1+\beta})}\right)^{k}\right] \quad (14)$$

The learning parameter illustrates how patch deployment improves over time. Initially, patch deployment progresses slowly as organisations address initial challenges, such as testing patches, verifying compatibility, and training staff. As time passes, the learning effect enhances deployment efficiency, leading to improved coordination, more effective patch rollouts, and increasingly automated deployment strategies.

4. MODEL VALIDATION

To validate the proposed modelling framework, the authors used a real-world patch dataset for three operating systems: Android, Windows 10, and RedHat.

This section outlines the primary processes for collecting the data required to evaluate the proposed modelling framework and to assess its goodness of fit. It also mentions various repositories representing multiple companies and software developers, which provide advisory reports on discovered vulnerabilities, including patches. The datasets used in this investigation were obtained from the following sources:

- Vendor advisories: Security reports, such as Microsoft Security Bulletins (MS) and Mozilla Firefox Security Advisories (MFSA), that list fixes for flaws discovered by software vendors.
- Advisories from third parties: The National Vulnerability Database (NVD) and CVE Details (www.cvedetails.com) are other resources that provide current information on vulnerabilities and patches.

The authors gathered vulnerability and patch release datasets for the same period for the investigation. Patch datasets were obtained from Red Hat (January 2020–April 2021), Windows 10 (January 2021-June 2022), and CVE details for Android (July 2021–December 2022). The parameters of the suggested modelling framework were estimated using these datasets, and its performance was assessed using a range of goodness-of-fit criteria.

4.1. Patch data extraction steps - Android (DS-I)

Step 1: First, visit the site www.cvedetails.com and click on the “Products” category, as shown in Figure 2. Then, use the search bar to find the product you are looking for. In this case, we’re searching for patches for Android. Select the number of vulnerabilities associated with the corresponding product.

Step 2: Choose the year and month according to the requirements. Here, we have chosen the vulnerabilities published in July 2021. The total number of vulnerabilities identified in July 2021 is shown in Figure 3.

Step 3: The next step is to find the number of patches released in July 2021. We click on every CVE ID to check whether a patch has been released for that vulnerability. For example, in Figure 4, it can be seen that a patch has been released for CVE-2021-0604.

The data was similarly extracted for Windows and Red Hat.

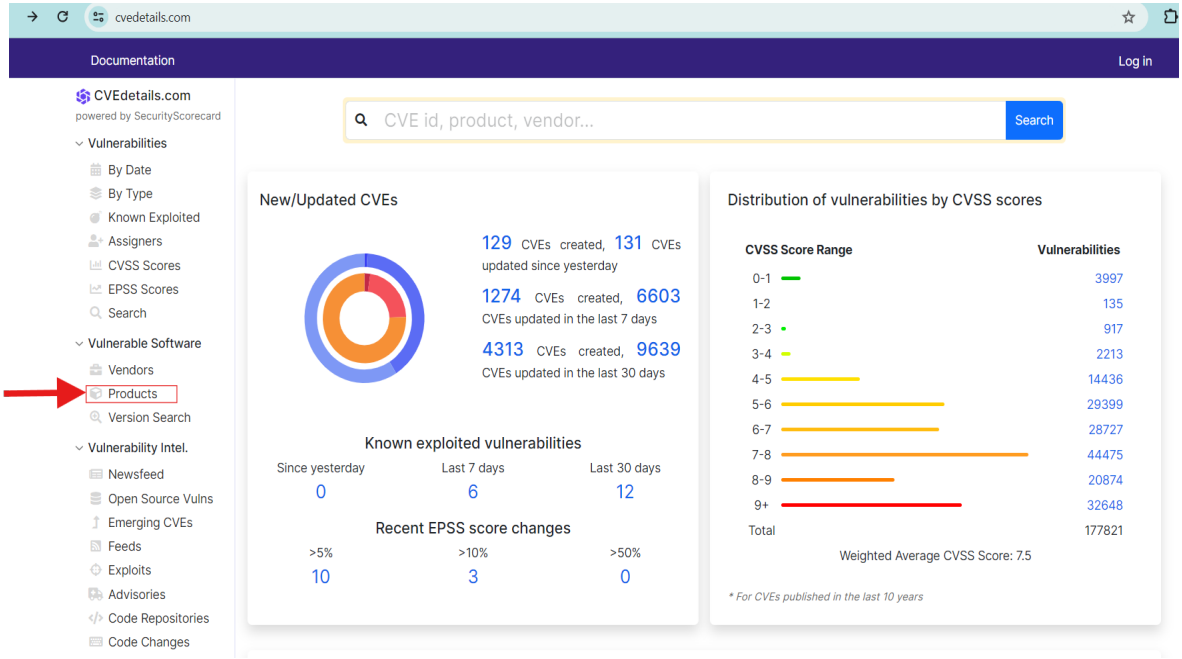

Figure 2: CVE details site

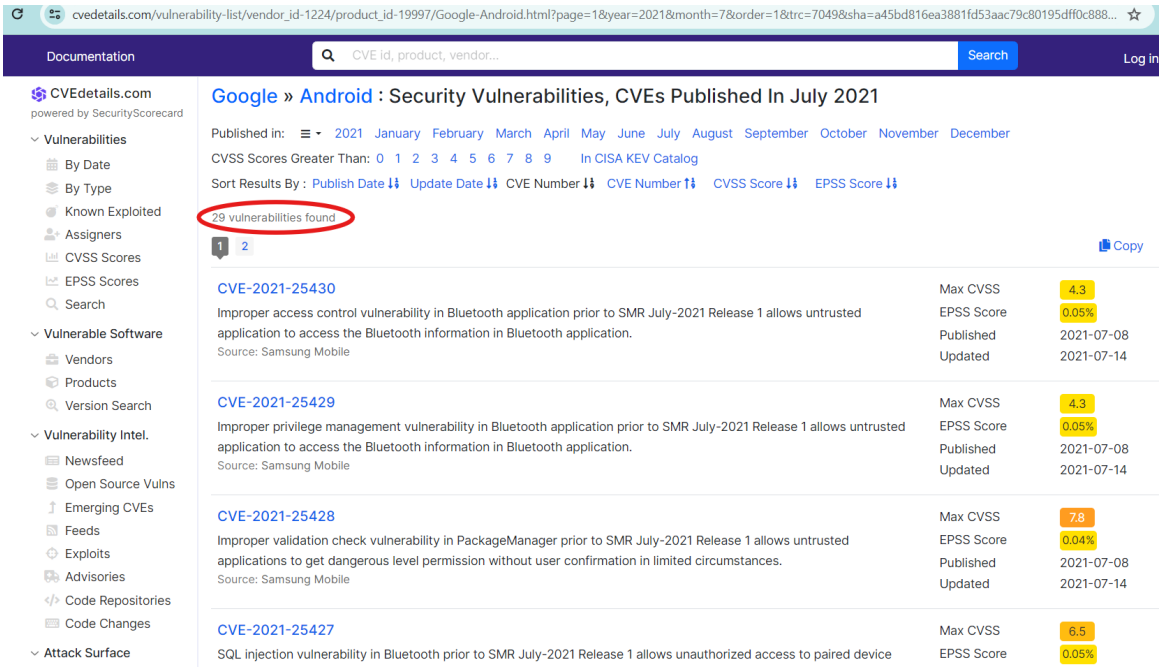

Figure 3: Total number of vulnerabilities published in July 2021 on cvedetails.com

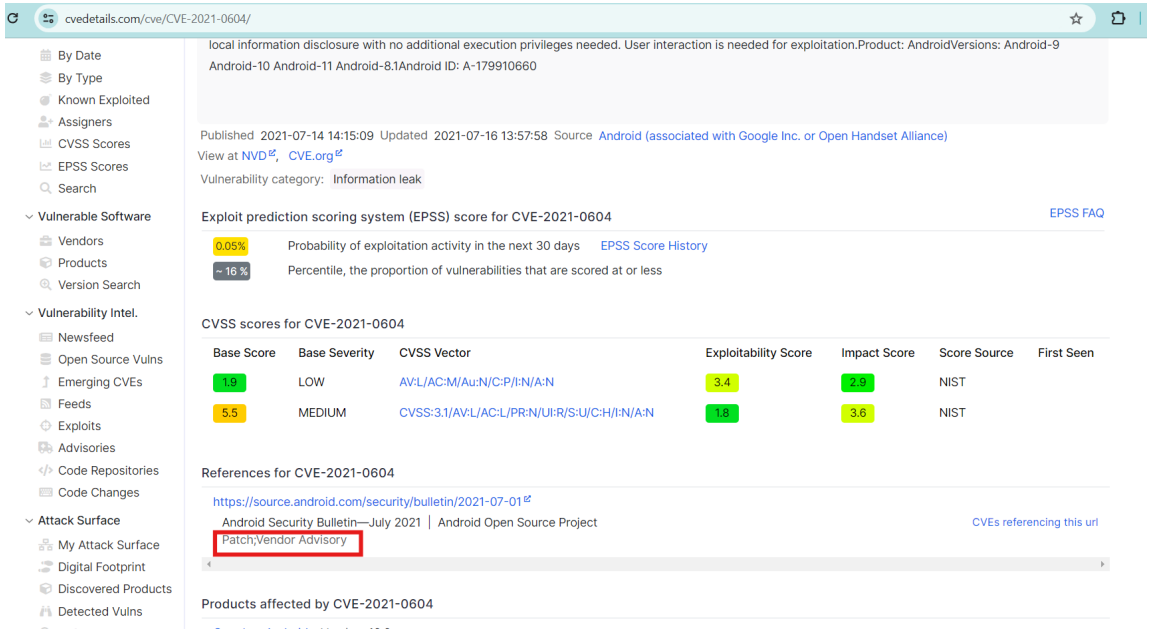

Figure 4: Patch released for CVE-2021-0604

The author computed the parameters for the suggested model using SPSS. The parameters of the proposed model were estimated using nonlinear least squares estimation (NLSE) implemented in SPSS. The statistical fit of a patch deployment model that takes uncertainty into account depends on various parameters. Factors such as R-squared, variance, bias, mean squared error (MSE), and mean absolute error (MAE) were used by the authors. When variance, bias, and MSE are minimised, the model closely matches the actual data. A higher R-squared indicates a stronger correlation between expected and observed data. Tables 2, 4, and 6 present parameter estimates for all datasets, clearly showing that Case 3, which includes a learning effect, produces the best results. This case involves a learning effect that gives the patching rate a variable lifecycle compared with Cases 1 and 2, allowing for more realistic modelling of patch deployments. By adding a learning parameter, Case 3 realistically prevents infinite growth by gradually increasing patching speed from slow to fast. The model naturally stabilises at optimal conditions. In addition to simulating a process that improves over time through learning, Case 3 also stabilises owing to constraints related to scheduling and resources. Unlike Case 1, which assumes a constant rate, and Case 2, which assumes indefinite linear patching, Case 3 captures the added complexity of patching that improves over time.

Table 2: Values of estimated parameters (DS-I)

Model parameter (Android) DS-I	A	λ	q	β	k
Case 1	4798.21	51.14	0.51	-	0.23
Case 2	94729.89	984.39	0.31	-	0.97
Case 3	5064.56	124.96	0.99	7.86	0.11

Table 3: Goodness-of-fit criteria for Android (DS-I)

Model parameter (Android) DS-I	Bias	MSE	Variance	MAE	R-square
Case 1	-10.82	812.13	6.50	20.85	0.99
Case 2	-1.67	224.13	3.42	10.83	0.97
Case 3	-0.32	190.25	3.16	8.84	0.99

Table 4: Values of estimated parameters (DS-II)

Model parameter (Windows) DS-II	A	λ	q	β	k
Case 1	11545.99	3.12	0.023	-	0.53
Case 2	6856.37	40.41	0.002	-	0.25
Case 3	1025.88	0.153	0.059	12.00	0.02

Table 5: Goodness-of-fit criteria for Windows (DS-II)

Model parameter (Windows) DS-II	Bias	MSE	Variance	MAE	R-square
Case 1	1.99	249.99	3.57	12.23	0.99
Case 2	7.81	1539.59	8.82	34.01	0.97
Case 3	1.98	246.77	3.55	12.23	0.99

Table 6: Values of estimated parameters (DS-III)

Model parameter (Redhat) DS-III	A	λ	q	β	k
Case 1	277.96	4.37	0.99	-	0.02
Case 2	360.63	1.48	0.04	-	0.52
Case 3	358.21	0.001	0.02	26.30	0.26

Table 7: Goodness-of-fit criteria for Red Hat (DS-III)

Model parameter (Redhat) DS-III	Bias	MSE	Variance	MAE	R-square
Case 1	2.81	353.28	5.45	353.28	0.94
Case 2	17.28	1214.04	7.44	554.35	0.90
Case 3	1.66	215.30	4.86	12.09	0.97

Tables 3,5, and 7 represent the comparison criteria for all three datasets. The proposed modelling approach is perfectly suited to the observed sample. Case 3 produces the best outcomes of the three cases. In addition, Figures 5, 6, and 7 for Dataset I, Dataset II, and Dataset III respectively show a close fit according to the R-squared value.

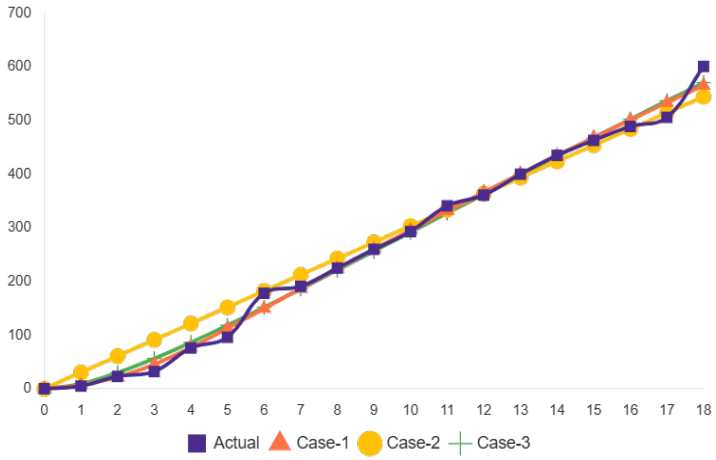

Figure 5: Goodness-of-fit curve for DS-I (Android)

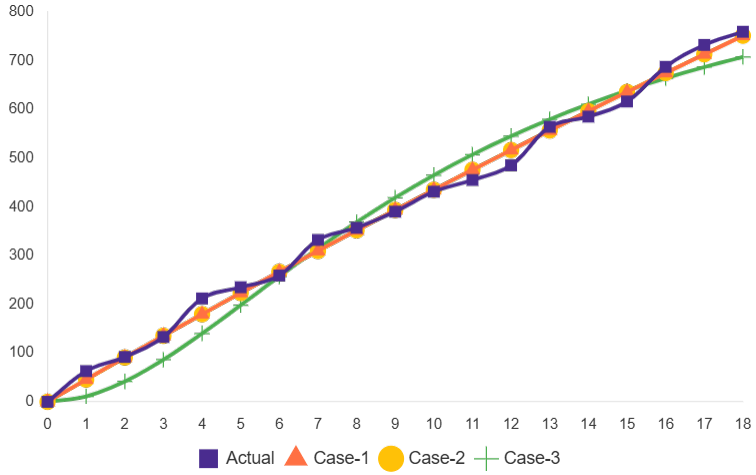

Figure 6: Goodness-of-fit curve for DS-II (Windows)

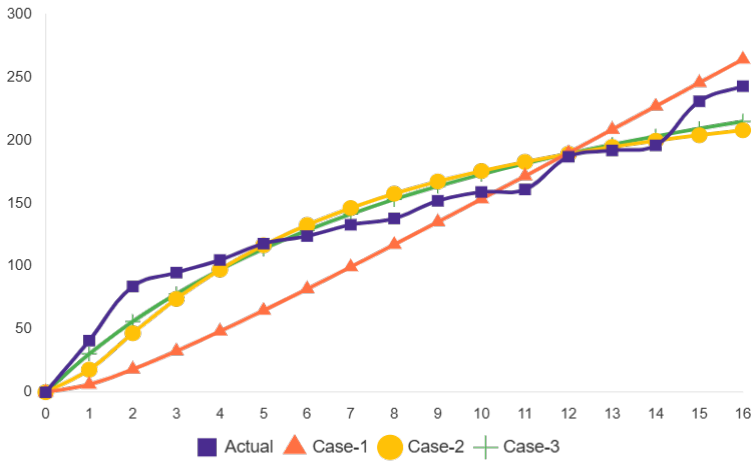

Figure 7: Goodness-of-fit curve for DS-III (Redhat)

The graphical analysis of the patching phenomenon between Datasets I, II, and III reveals that the model accurately replicates actual patching trends over time. The authors have also confirmed the model's ability to describe software patching behaviour, as actual and predicted values matched with high accuracy from both individual and cumulative distribution perspectives. The close match between actual and modelled values indicates that the model can successfully forecast patch success, which the authors believe will improve software vulnerability management and overall software security strategies. The visual validation adds to evidence of the model's validity, complementing goodness-of-fit and statistical measures.

The validation of the proposed framework is carried out through comparative analysis of three different cases derived from the same model structure. Each case represents a distinct functional form of the patch deployment rate $q(t)$, capturing different real-world deployment behaviours such as constant rate, nonlinear growth, and learning-driven dynamics. The objective of this comparison is not merely internal validation, but also to examine the robustness and adaptability of the proposed modelling framework under varying operational conditions. Since all three cases are derived from the same theoretical foundation incorporating environmental uncertainty and stochastic effects, differences in performance between these cases highlight the sensitivity of the model to deployment dynamics.

In addition, the proposed framework enhances traditional NHPP-based models by explicitly including environmental uncertainty as a random variable and incorporating learning effects into the deployment process. As a result, the validation emphasises illustrating how these new components affect model behaviour in various scenarios, rather than comparing it with structurally different models.

5. RESULTS AND DISCUSSION

Real-world patching datasets for Red Hat, Windows 10, and Android operating systems were used to validate the proposed mathematical modelling approach. Three scenarios – constant deployment rate, deployment affected by quadratic growth, and deployment with a learning parameter – served as the basis for the comparative analysis. The model validation results, particularly the goodness-of-fit metrics, such as bias, MSE, variance, MAE, and R-squared values, show that Case 3 (deployment rate with a learning parameter) outperforms the other two cases. This highlights the significance of the learning parameter in improving the model's adaptability to real-world patching dynamics. The findings from Tables 3, 5, and 7 indicate the following:

- Android (DS-I): Case 3 provides the best fit, with the lowest bias (-0.32), the lowest MSE (190.25), and the highest R-squared (0.99) indicating that the patch deployment process is influenced by learning effects.
- Windows (DS-II): Case 3 again demonstrates a superior fit with minimal bias (1.98), low MSE (246.77), and high R-squared (0.99), validating the model's robustness in different OS environments and reassuring the audience about its broad applicability.
- Red Hat (DS-III): The model successfully captures patching uncertainty, with Case 3 showing better results (Bias = 1.66, MSE = 215.30, R-squared = 0.97) than Cases 1 and 2. This underscores the model's ability to capture patching uncertainty, thereby enhancing its predictive accuracy.

Introducing environmental factors as random variables into the model accurately describes real-world uncertainties, thereby ensuring greater accuracy in predicting patch success rates. Case 3, which includes the learning factor, provides a good model of dynamic patching procedures, acknowledging the gradual rise in efficiency while considering the initial slow acceptance of the patch. The organisation that undertakes these studies could use this information to prioritise resources and strategies for patch management more effectively, reducing vulnerabilities and offering hope for a more secure digital landscape.

6. LIMITATIONS AND FUTURE SCOPE

Future studies may address some limitations of the proposed model, despite its improved description of patch deployment in uncertain situations. The model assumes that fixes are applied immediately after vulnerabilities are identified, which may not always occur in real life owing to administrative permissions and testing delays. It cannot capture long-term changes in operational conditions because the variables affecting patch deployment are treated as random, rather than as variables that change over time. The model is validated with three operating system datasets; however, validation with different software ecosystems, applications, and industries is also necessary. The framework does not explicitly recognise administrative policies and human factors in decision-making; instead, it focuses on systematic and automated patching methods.

Future research could improve the model in several ways. It could consider environmental factors that affect time-varying variables, which rely on past data and change in response to security needs. There may be gains in predictive ability by examining trends in patching behaviour over time, and using machine learning techniques within the proposed mathematical framework. In addition, broadening this framework to include other contexts, such as cloud computing and Internet of Things devices, could clarify security patching in distributed and resource-limited environments. Ultimately, this research may prompt a shift toward a real-time monitoring system based on the framework, which would enhance the organisation's cybersecurity resilience by dynamically adapting patch deployment methods.

7. CONCLUSION

This study has demonstrated a new mathematical modelling method that integrates the idea of environmental uncertainties into software patch management. The modelling attempts to demonstrate the real-world behaviour of patch deployment by integrating the environmental uncertainty with a non-homogeneous Poisson process. The proposed model was validated using actual patching data from Red Hat, Windows 10, and Android, and it outperforms existing traditional constant-rate models. The key findings highlight the role of learning effects in the patch deployment process. For example, the results of Case 3 are the best predictive estimates. Therefore, this model has provided valuable insights for a rapidly changing operating environment, improving the effectiveness of security patching methods while increasing resilience against remaining vulnerabilities and enhancing the reliability of the systems. Future work would

involve incorporating aspects of machine learning, dynamic modelling of environmental attributes, and applications to newer technologies, such as cloud computing and the Internet of Things (IoT), and their influence on this framework. Overall, this study makes a significant contribution to cybersecurity by providing a robust method to address uncertainty in the context of software patch management, thereby enhancing vulnerability mitigation strategies in practice.

REFERENCES

[1] **Choi, T. M., Dolgui, A., Ivanov, D., & Pesch, E.** (2022). OR and analytics for digital, resilient, and sustainable manufacturing 4.0. *Annals of Operations Research*, *310*(1), 1-6.

[2] **Roland Weistroffer, H., & Narula, S. C.** (1997). The state of multiple criteria decision support software. *Annals of Operations Research*, *72*(0), 299-313.

[3] **McGraw, G.** (2004). Software security. *IEEE Security & Privacy*, 2(2), 80-83.

[4] **Wu, Y., Jiang, N., Pham, H. V., Lutellier, T., Davis, J., Tan, L., Babkin, P., & Shah, S.** (2023, July). How effective are neural networks for fixing security vulnerabilities? In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 1282-1294).

[5] **Bhatt, N., Anand, A., & Yadavalli, V. S.** (2021). Exploitability prediction of software vulnerabilities. *Quality and Reliability Engineering International*, *37*(2), 648-663.

[6] **Suri, R.** (1985). An overview of evaluative models for flexible manufacturing systems. *Annals of Operations Research*, *3*(1), 13-21.

[7] **Dissanayake, N., Jayatilaka, A., Zahedi, M., & Babar, M. A.** (2022). Software security patch management: A systematic literature review of challenges, approaches, tools and practices. *Information and Software Technology*, *144*, 106771.

[8] **Rescorla, E**. (2005). Is finding security holes a good idea? *IEEE Security & Privacy*, *3*(1), 14-19.

[9] **Anand, A., Bhatt, N., & Alhazmi, O. H.** (2021). Vulnerability discovery modelling: A general framework. *International Journal of Information and Computer Security*, *16*(1-2), 192-206.

[10] **Alhazmi, O. H., & Malaiya, Y. K.** (2008). Application of vulnerability discovery models to major operating systems. *IEEE Transactions on Reliability*. *57*(1), 14-22.

[11] **Massacci, F., & Nguyen, V. H.** (2014). An empirical methodology to evaluate vulnerability discovery models. *IEEE Transactions on Software Engineering*, *40*(12), 1147-1162.

[12] **Kaur, J., Anand, A., & Singh, O.** (2019). Modeling software vulnerability correction/fixation process incorporating time lag. In *Recent Advancements in Software Reliability Assurance*. CRC Press (pp. 39-58).

[13] **Movahedi, Y., Cukier, M., & Gashi, I.** (2019). Vulnerability prediction capability: A comparison between vulnerability discovery models and neural network models. *Computers & Security*, *87*, 101596.

[14] **Liu, Q., & Xing, L.** (2021). Survivability and vulnerability analysis of cloud RAID systems under disk faults and attacks. *International Journal of Mathematical, Engineering and Management Sciences*, *6*(1), 15-29.

[15] **Nappa, A., Johnson, R., Bilge, L., Caballero, J., & Dumitras, T.** (2015, May). The attack of the clones: A study of the impact of shared code on vulnerability patching. In *2015 IEEE symposium on security and privacy*. IEEE, (pp. 692-708).

[16] **Anand, A., & Bhatt, N.** (2016). Vulnerability discovery modeling and weighted criteria-based ranking. *Journal of the Indian Society for Probability and Statistics*, *17*, 1-10.

[17] **Sharma, R., Sibal, R., & Shrivastava, A. K.** (2016). Vulnerability discovery modeling for open and closed source software. *International Journal of Secure Software Engineering (IJSSE)*, *7*(4), 19-38.

[18] **Bhatt, N., Anand, A., Yadavalli, V. S. S., & Kumar, V**. (2017). Modeling and characterizing software vulnerabilities. *International Journal of Mathematical, Engineering and Management Sciences*, *2*(4), 288-299. https://dx.doi.org/10.33889/IJMEMS.2017.2.4-022

[19] **Anand, A., Das, S., Aggrawal, D., & Klochkov, Y.** (2017). Vulnerability discovery modelling for software with multi-versions. In M. Ram, & J. Paulo Davim, eds, *Advances in reliability and system engineering*. Springer, Cham (pp. 255-265).

[20] **Li, F., & Paxson, V.** (2017, October). A large-scale empirical study of security patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (pp. 2201-2215).

[21] **Anand, A., Agarwal, M., Tamura, Y., & Yamada, S**. (2017). Economic impact of software patching and optimal release scheduling. *Quality and Reliability Engineering International*, *33*(1), 149-157.

[22] **Shrivastava, A. K., & Sharma, R.** (2019). Modeling vulnerability discovery and patching with fixing lag. In *Advanced Informatics for Computing Research: Second International Conference, ICAICR 2018*, Part II, 2, Springer Singapore, (pp. 569-578).

[23] **Almukaynizi, M., Nunes, E., Dharaiya, K., Senguttuvan, M., Shakarian, J., & Shakarian, P**. (2019). Patch before exploited: An approach to identify targeted software vulnerabilities. In Sikos, L. (ed), *AI in cybersecurity*. Intelligent Systems Reference Library, vol 151. Springer, Cham (pp. 81-113).

[24] **Anand, A., Agrawal, M., Bhatt, N., & Ram, M.** (2019). Software patch scheduling policy incorporating functional safety standards. In M. Ram, & J. Paulo Davim, eds, *Advances in system reliability engineering*. Academic Press (pp. 267-279).

[25] **Anjum, M., Kapur, P. K., Agarwal, V., & Khatri, S. K.** (2020). Assessment of software vulnerabilities using best-worst method and two-way analysis. *International Journal of Mathematical, Engineering and Management Sciences*, *5*(2), 328-342.

[26] **Kansal, Y., Kumar, D., & Kapur, P. K.** (2016). Vulnerability patch modeling. *International Journal of Reliability, Quality and Safety Engineering, 23*(06), 1640013.

[27] **Anand, A., Gupta, P., Klochkov, Y., & Yadavalli, V. S**. (2018). Modeling software fault removal and vulnerability detection and related patch release policy. In A. Anand & M. Ram, eds, e*System reliability management*. CRC Press (pp. 19-34)

[28] **Shrivastava, A. K., Kapur, P. K., & Anjum, M.** (2019). Vulnerability discovery and patch modeling: State of the art. *Reliability Engineering, 14*, 401-419.

[29] **Anand, A., Bhatt, N., & Aggrawal, D.** (2020). Modeling software patch management based on vulnerabilities discovered. *International Journal of Reliability, Quality and Safety Engineering*, *27*(02), 2040003.

[30] **Aggrawal, D., Kaur, J., & Anand, A.** (2022). Modeling software patching process inculcating the impact of vulnerabilities discovered and disclosed. In P. Johri, A. Anand, J. Vain, J. Singh, & M. Quasim, eds, *System assurances: Modeling and management*. Academic Press (pp. 143-153).

[31] **Divya, A. A., Bhatt, N., & Johri, P.** (2024). Assessing the Impact of software patching on vulnerabilities: A comprehensive framework for faulty and safe patches. *International Journal of Reliability, Quality and Safety Engineering, 32*(2), 2450031.

[32] **Anand, A., Kaur, J., & Inoue, S.** (2020). Reliability modeling of multi-version software system incorporating the impact of infected patching. *International Journal of Quality & Reliability Management*, *37*(6/7), 1071-1085.

[33] **Bhatt, N., Kaur, J., Anand, A., & Alhazmi, O. H.** (2022). Selecting best software vulnerability scanner using intuitionistic fuzzy set TOPSIS. *Computers, Materials & Continua*, *72*(2), 3613-3629.

[34] **Anand, A., Divya, A., Aggrawal, D., & Alhazmi, O. H.** (2025). Multi-phase modeling for vulnerability detection & patch management: An analysis using numerical methods. *Computers, Materials & Continua, 84*(1), 1529-1544. https://doi.org/10.32604/cmc.2025.063361

[35] **Teng, X., & Pham, H.** (2006). A new methodology for predicting software reliability in the random field environments. *IEEE Transactions on Reliability*, *55*(3), 458-468.

[36] **Chang, I. H., Pham, H., Lee, S. W., & Song, K. Y.** (2014). A testing-coverage software reliability model with the uncertainty of operating environments. *International Journal of Systems Science: Operations & Logistics*, *1*(4), 220-227.

[37] **Pham, H.** (2014). A new software reliability model with Vtub-shaped fault-detection rate and the uncertainty of operating environments. *Optimization*, *63*(10), 1481-1490.

[38] **Pham, H.** (2016). A generalized fault-detection software reliability model subject to random operating environments. *Vietnam Journal of Computer Science*, *3*(3), 145-150.

[39] **Li, Q., & Pham, H.** (2017). NHPP software reliability model considering the uncertainty of operating environments with imperfect debugging and testing coverage. *Applied Mathematical Modelling, 51*, 68-85.

[40] **Zhu, M., & Pham, H.** (2018). A software reliability model incorporating martingale process with gamma-distributed environmental factors. *Annals of Operations Research, 354(*3), 1389-1410.

[41] **Li, Q., & Pham, H.** (2019). A generalized software reliability growth model with consideration of the uncertainty of operating environments. *IEEE Access*, *7*, 84253-84267