Quantifying Cross-Release Vulnerability Discovery in Software: A Multi-Release Modeling Approach

Jyotish N. P. Singh[1], Navneet Bhatt[2], Adarsh Anand[3,*] and Divya[3]

[1]*NICMAR University, Pune, Maharashtra, India*
[2]*Anil Surendra Modi School of Commerce, SVKM's Narsee Monjee Institute of Management Studies (NMIMS), Mumbai 400056, India*
[3]*Department of Operational Research, University of Delhi, Delhi 110007, India*
E-mail: jyotish.singh@pune.nicmar.ac.in; navneetbhatt@live.com; adarsh.anand86@gmail.com; divya@or.du.ac.in
[*]*Corresponding Author*

Received 12 September 2025; Accepted 29 November 2025

Abstract

Software Vulnerability Discovery Models (VDMs) help understand how security flaws are identified in software after release. Traditionally, most models focus on a single release and assume vulnerabilities are discovered independently. However, modern software often exists in multiple versions that share a significant portion of code. This shared structure means that discovering a vulnerability in one version can also help identify similar issues in other versions: a phenomenon referred to as cross-discovery. In this paper, the author extends the traditional VDM framework to model multiple software releases simultaneously, enabling interaction between versions via cross-discovery. Unlike existing multi-release models that use constant values

Journal of Graphic Era University, Vol. 14_1, 183–204.
doi: 10.13052/jgeu0975-1416.1416

for shared-code influence, the author proposes a new model that introduces time-dependent cross-discovery coefficients. These functions reflect how the influence between software versions changes over time, making the model more realistic.

The author applies the proposed model to real-world data from two popular Windows operating system versions, Windows XP and Windows Vista. The proposed model accurately captures both within-release discoveries and those triggered by interaction with the other version. The author estimates parameters, validates results using statistical measures, and visualizes predicted and actual trends. The results show that the proposed approach provides deeper insights into how vulnerabilities are found across multiple software releases, supporting better software maintenance and security planning.

Keywords: Multi-release software, patch, security, vulnerability discovery models (VDMs).

1 Introduction

The rapid growth of digital technologies and the increasing reliance on software systems have significantly raised concerns around software security. Despite frequent upgrades and the addition of new features, software applications often continue to contain flaws. These flaws, introduced accidentally during the software development lifecycle (SDLC), are known as *software vulnerabilities*. A widely accepted definition by Krsul [1] describes a vulnerability as "an instance of a mistake in the specification, development, or implementation of a software such that its execution may violate the security policy." These vulnerabilities can be discovered during the software's operational phase and need to be patched quickly to avoid exploitation.

Vulnerabilities go through a life cycle that includes several stages: injection, discovery, disclosure, public announcement, patching, and potential exploitation. During the operational phase, vulnerabilities are discovered at different rates depending on user activity, attention from attackers or security researchers, and the availability of newer software versions. Historical cases such as the Code Red worm and the Slammer worm demonstrate how quickly vulnerabilities can be exploited, often within minutes of discovery, leading to widespread disruption.

Understanding and predicting the discovery of software vulnerabilities is crucial. To do this, researchers have developed Vulnerability Discovery

Models (VDMs) inspired by earlier Software Reliability Growth Models (SRGMs). While SRGMs focus on faults detected during testing before software release, VDMs model the vulnerabilities found after release during actual usage. However, traditional VDMs often study a single software release in isolation and do not account for the fact that multiple software versions are usually supported and used concurrently.

Modern software products typically evolve through multiple releases, where newer versions retain a significant portion of code from older versions. For example, a vulnerability discovered in one version of a system may also exist in its predecessor or successor due to shared code. This leads to a phenomenon known as *cross-discovery*, where discovering a vulnerability in one release helps uncover similar vulnerabilities in other releases. For instance, the vulnerability CVE-2011-5046 discovered in Windows 7 x64 was also present in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 [2].

This study introduces a new multi-release Vulnerability Discovery Model (VDM) that reflects the actual process of cross-discovery between different software versions. In contrast to most existing models that assume a consistent influence from shared code, the proposed model considers this influence as a function that varies over time. This change offers a more accurate perspective, since the impact of one release on another often fluctuates, either diminishing as older versions fade in popularity or growing stronger due to increased shared code analysis.

The author begins in Section 2 by discussing related work on vulnerability discovery and software reliability modeling, emphasizing the limitations of current models in handling multiple software versions. Section 3 introduces the proposed multi-release vulnerability modeling framework, which extends traditional approaches by incorporating time-dependent cross-discovery functions to reflect real-world software dynamics better. In Section 4, the author describes the dataset used for model application, focusing on vulnerability data from Windows XP and Windows Vista. Section 5 outlines the evaluation metrics used to assess model performance, including Mean Squared Error (MSE), Akaike Information Criterion (AIC), Relative Mean Percentage Square Error (RMPSE), and the coefficient of determination (R^2). The authors then present the results of parameter estimation in Section 6, including predictive analysis and model fit. Section 7 explores the cross-discovery interactions between the two releases in more detail, using both tabular and graphical representations. Finally, Section 8 concludes the study with key findings, implications, and directions for future research.

2 Related Work

The goal of vulnerability discovery models, or VDMs, is to characterize and predict the number of software flaws that will be found over time for a product or family of products. Forecasting aids operators and vendors in prioritizing security investments, planning patching, allocating resources for testing and inspection, and estimating exposure windows. VDMs can be time-based (calendar time), effort-based (testing effort, usage), or multivariate (including covariates like reporters, code reuse, and market share).

Over the years, numerous researchers have contributed to understanding the process of software vulnerability discovery, leading to the development of various Vulnerability Discovery Models (VDMs). These models are inspired by Software Reliability Growth Models (SRGMs), but in comparison to SRGMs, which focus on pre-release fault detection, VDMs are designed to model the discovery of security flaws after a software product has been released.

The foundational work in this area was done by Rescorla [3], who conducted a cost-benefit analysis of vulnerability disclosure, emphasizing that discovering a vulnerability by a "good actor" is less harmful than if found by a malicious one. He also analyzed vulnerability trends in several operating systems to determine whether the discovery patterns followed linear or exponential behavior. Similarly, Roger Needham [4] highlighted the unique nature of security bugs compared to regular software bugs, noting that malicious intent and external attackers make security vulnerabilities particularly critical.

A widely cited framework by Alhazmi and Malaiya [5, 7] proposed that the vulnerability discovery process evolves over time and is influenced by user engagement and familiarity. Initially, users need time to understand a system, resulting in fewer discoveries. As understanding improves, the discovery rate increases rapidly until it reaches a saturation phase, not necessarily because all vulnerabilities are found, but because user interest declines, often due to the availability of newer software versions. This behavior was effectively modeled using a logistic S-shaped curve, and further validated across various software types, from operating systems to browsers [6].

Building on this foundation, several researchers proposed models to capture more specific characteristics of vulnerability discovery. Joh et al. [8] introduced an asymmetric discovery model using the Weibull distribution, while Younis et al. [9] employed a folded normal distribution for systems with minimal learning phase. Joh and Malaiya [10] explored the impact

of data skewness, and Kapur et al. [11, 22] proposed a learning-based S-shaped model to represent vendor efficiency in discovering vulnerabilities. Anand and Bhatt [12] examined market-driven discovery patterns, introducing a "hump-shaped" model to reflect changing user interest along with that the paper helps in prioritize testing and remediation by creating ranking/decision frameworks that include vulnerability discovery model outputs with weighted criteria. This work connects decision-analytic ranking techniques with traditional VDM outputs. Bhatt et al. [13] further investigated the influence of already discovered vulnerabilities on subsequent discoveries. The paper offers empirical descriptions that are helpful for later VDM fitting and interpretation. It also presents characterizations of vulnerability data and preliminary modeling work on how vulnerabilities appear in software systems. Another empirical work proposed by Bhatt et al. [14] connects model forecasts to operational decisions by optimizing resource allocation for vulnerability discovery and testing using VDM outputs and with was followed by reducing risk while taking operational constraints into account by incorporating VDM-derived discovery curves into patch scheduling and management policies [15].

Along in lines, Anand et al. [16] proposed a vulnerability discovery model to account for discovery incentives and reporter-driven discovery dynamics, it goes beyond VDMs by explicitly modeling the role of reporters, who find and reveal vulnerabilities. This is a crucial step toward more realistic, multivariate VDMs. Anand et al. [17] later proposed a general VDM framework that unifies various VDMs and offers recommendations for model selection and evaluation across datasets. Researchers can select suitable VDM forms and validation techniques using this framework. Anand et al. [18] proposed a multi-phase modelling for vulnerability detection and patch management using numerical methods. Divya et al. [19] proposed a modelling framework to assess the impact of software patching on vulnerabilities for both faulty and safe patching.

Despite these advancements, most existing VDMs focus on a single software release and do not account for the interaction between co-existing versions. However, in practice, software vendors maintain and support multiple releases simultaneously, and these versions often share a significant portion of the codebase. The presence of shared code implies that a vulnerability found in one version might also exist and be identified in another. This interlinked behavior, which is termed cross-discovery, plays a critical role in shaping vulnerability trends across versions.

A limited number of studies have attempted to address the multi-release scenario. Kim et al. [20] proposed a framework extending the AML model, where vulnerabilities shared across versions are inherited by newer releases. Later, Anand et al. [21] introduced a mathematical model where some vulnerabilities, although not detected in a prior version, were discovered later in newer versions and subsequently mapped back to older releases. These approaches recognize the importance of inter-version dependency but largely rely on constant coefficients to represent shared-code influence.

The proposed work extends this line of research by introducing a more flexible and realistic modeling approach that allows the influence of shared code to vary over time. Specifically, the author proposes time-dependent cross-discovery coefficients that capture how the influence of discovery between versions changes dynamically, such as increasing during overlapping usage or decreasing as legacy versions become obsolete. This enhancement offers a more accurate understanding of real-world vulnerability trends in multi-release environments and aligns with how modern software ecosystems evolve.

In contrast to these earlier studies, which employ constant coefficients to model inter-version dependencies, our approach introduces a time-varying formulation. This enables a more realistic representation of how the discovery influence evolves as software matures across releases. Thus, the proposed model can be viewed as an extension and generalization of the frameworks proposed by Kim et al. [20] and Anand et al. [21], serving as baseline formulations for comparison in our numerical evaluation.

3 Modeling Framework

This section presents a time-dependent multi-release Vulnerability Discovery Model (VDM) that captures the dynamics of vulnerability discovery in multiple coexisting software versions. The model is designed to reflect the reality of shared codebases, varying discovery rates, and cross-discovery, where discovering a vulnerability in one release may lead to detecting the same flaw in other versions. The framework builds upon the logistic S-shaped model previously adopted in vulnerability modeling studies [5, 11, 13] and extends it by incorporating dynamic cross-discovery behavior.

3.1 Assumptions

1. Multiple versions of a software coexist after successive releases.

2. Each version introduces new features and retains code from previous versions.
3. Vulnerabilities may exist across multiple versions due to code reuse.
4. The discovery of a vulnerability in one version may lead to its detection in another; this is modelled through a time-dependent cross-discovery coefficient.

3.2 Model for a Single Release

Considering the situation when only first release of a software is available in the market and it is released at time t_0, then the number of vulnerabilities discovered in the time period $t_0(=0) \leq t \leq t_1$ can be modeled as:

$$\Omega_1(t) = N_1 \cdot F_1(t - t_0) \tag{1}$$

Where:

- $\Omega_1(t)$: Cumulative number of vulnerabilities discovered in Release 1 by time ttt
- N_1: Total number of vulnerabilities in Release 1
- β_1: Shape parameter
- r_1: Vulnerability discovery rate for Release 1
- $F_1(t - t_0)$: Logistic Vulnerability discovery function for release 1, given by

$$F_1(t - t_0) = \left(\frac{1 - e^{-r_1(t-t_0)}}{1 + \beta_1 \cdot e^{-r_1(t-t_0)}} \right)$$

3.3 Modeling for Two Coexisting Releases

Let the second software release (Release 2) be launched at time t_1, where $t_1 > t_0$. From this point onward, both versions coexist in the market. Vulnerabilities can be discovered independently in each version or identified through cross-discovery.

For time $t > t_1$, the cumulative vulnerabilities discovered in both releases are modeled as:

For Release 1:

$$\Omega_1(t) = N_1 \cdot F_1(t_1) + N_1 \cdot F_1(t - t_1) + \alpha_{12}(t) \cdot N_2 \cdot F_2(t - t_1) \tag{2}$$

For Release 2:

$$\Omega_2(t) = N_2 \cdot F_2(t - t_1) + \alpha_{21}(t) \cdot N_1 \cdot F_1(t - t_1) \tag{3}$$

Where:

- $\Omega_2(t)$: Cumulative number of vulnerabilities discovered in Release 2
- N_2: Total number of vulnerabilities in Release 2
- r_2: Vulnerability discovery rate for Release 2
- β_2: Shape parameter for Release 2
- $F_2(t - t_1)$: Logistic discovery function for Release 2, given by

$$F_2(t - t_1) = \left(\frac{1 - e^{-r_2(t-t_1)}}{1 + \beta_2 \cdot e^{-r_2(t-t_1)}} \right)$$

- $\alpha_{12}(t)$: Time-dependent proportion of vulnerabilities discovered in Release 2 that are also found in Release 1
- $\alpha_{21}(t)$: Time-dependent proportion of vulnerabilities discovered in Release 1 that are also found in Release 2

The functions $\alpha_{12}(t)$ and $\alpha_{21}(t)$ are time-varying to account for evolving similarity and support across versions. Their mathematical forms can be modeled using an exponential decay function:

$$\alpha_{ij}(t) = \lambda_{ij}\, e^{-\gamma_{ij}(t-t_1)}$$

Where:

- λ_{ij}: Initial strength of cross-discovery from version j to version i
- γ_{ij}: Decay rate of cross-discovery influence

When a new release is introduced (e.g., Vista), it often shares much of its codebase with the older release (e.g., XP). This leads to strong initial cross-discovery effects. However, over time, differences increase due to patches, divergence in code evolution, and reduced attention to legacy systems, hence the influence decays.

Similarly, the author generalizes the described model for the expected number of vulnerabilities discovered till n-releases of the software given as:

$$\Omega_i(t) = N_i \cdot F_i(t_{n-1}) + N_i \cdot F_i(t - t_{n-1}) + \sum_{j=1(i\neq j)}^{n} \alpha_{ij}(t) \cdot N_j \cdot F_j(t - t_{n-1}); \quad i = 1, 2, \ldots n \tag{4}$$

The model reflects a real-world scenario in which two or more software versions share underlying components. A vulnerability found in one version may indicate the same issue in another. The functions $\alpha_{12}(t)$ and $\alpha_{21}(t)$ ensure that these cross-version influences are incorporated dynamically.

Table 1 Summary of vulnerabilities in Windows XP and Windows Vista

Year	Windows XP Vulnerabilities	Windows Vista Vulnerabilities
2001	6	–
2002	34	–
2003	45	–
2004	67	–
2005	89	–
2006	102	5
2007	115	29
2008	97	33
2009	81	40
2010	65	34
2011	42	22
2012	31	18
2013	24	15
2014	17	9
2015	11	7
2016	6	4
2017	2	1
Total	**834**	**247**

4 Data Analysis

To evaluate the applicability and effectiveness of the proposed multi-version vulnerability discovery model, the authors utilize real-world vulnerability data from two widely used Microsoft operating systems: Windows XP [23] and Windows Vista [24]. These operating systems represent sequential releases with overlapping market lifespans and codebase similarities, making them suitable for multi-version modeling and cross-discovery analysis. The vulnerability data is collected from the National Vulnerability Database (NVD), which provides a publicly accessible repository of reported software vulnerabilities.

5 Evaluation Metrics & Parameter Evaluation

To assess the performance and predictive accuracy of the proposed multi-version vulnerability discovery model, the author employs four widely accepted statistical evaluation metrics: Mean Squared Error (MSE), Akaike Information Criterion (AIC), Relative Mean Percentage Square Error (RMPSE), and the coefficient of determination (R^2). These metrics offer

Table 2 Parameter estimation for single release vulnerability discovery model

Parameter	Estimated Value
N_1	455.60
r_1	0.5247
β_1	20.04

Table 3 Estimated parameter values for the multi-release vulnerability discovery model

Parameter	Estimated Value
Release 1 (Windows XP)	
N_1	5022.43
r_1	0.1299
β_1	111.90
Release 2 (Windows Vista)	
N_2	138.69
r_2	0.7868
β_2	28.05
Cross-Discovery Parameters	
λ_{12}	4.6578
γ_{12}	0.0505
λ_{21}	5.0191
γ_{21}	0.0202

comprehensive insights into the model's fit, complexity, and forecasting capabilities.

The author presents the parameter estimation results for the proposed multi-version vulnerability discovery model using the dataset described in Section 4. The model was applied to vulnerability data from Windows XP (Release 1) and Windows Vista (Release 2).

Table 2 presents the estimated parameter values for Release 1 (Windows XP), which define the logistic vulnerability discovery curve when only the first release is available in the market.

Here is Table 3 showing the estimated parameter values for the proposed model considering both Release 1 (Windows XP) and Release 2 (Windows Vista), capturing both internal discovery and time-dependent cross-discovery effects.

Figure 1 presents the comparison between the actual and predicted cumulative vulnerabilities for Windows XP (Release 1) and Windows Vista (Release 2) over time, as captured by the proposed multi-release vulnerability discovery model. The proposed model effectively predicts the growth pattern

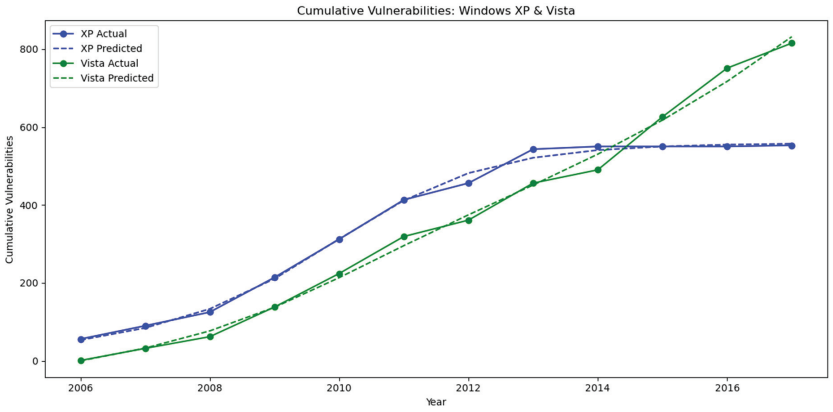

Figure 1 Actual vs. predicted cumulative vulnerabilities for Windows XP and Windows Vista.

Table 4 Model evaluation metrics for Windows XP and Windows Vista

Metric	Windows XP	Windows Vista
MSE	115.5761	351.7486
RMSE	10.7506	18.7550
RMPSE (%)	3.6561	29.9466
R^2	0.9968	0.9950
AIC	76.9992	90.3550

of Vista's vulnerabilities, even capturing the slight acceleration observed between 2015 and 2016.

Overall, the close alignment between the actual and predicted curves for both releases demonstrate the effectiveness of the proposed multi-release vulnerability discovery model in capturing both the individual release dynamics and the cross-discovery influence between successive software versions.

The performance metrics (Table 4 & Figure 2) provide a clear comparison between the vulnerability discovery models for Windows XP and Windows Vista. The **Mean Squared Error (MSE)** and **Root Mean Squared Error (RMSE)** are both lower for Windows XP, indicating that its model achieves higher prediction accuracy compared to the Vista model. Similarly, the **Relative Mean Percentage Squared Error (RMPSE)** is significantly smaller for XP, suggesting that the Vista model experiences larger relative prediction errors. Both models demonstrate strong predictive power, as reflected in their $\mathbf{R^2}$ values; however, the XP model performs marginally better in capturing the variance in the data. Finally, the **Akaike Information Criterion (AIC)** is also

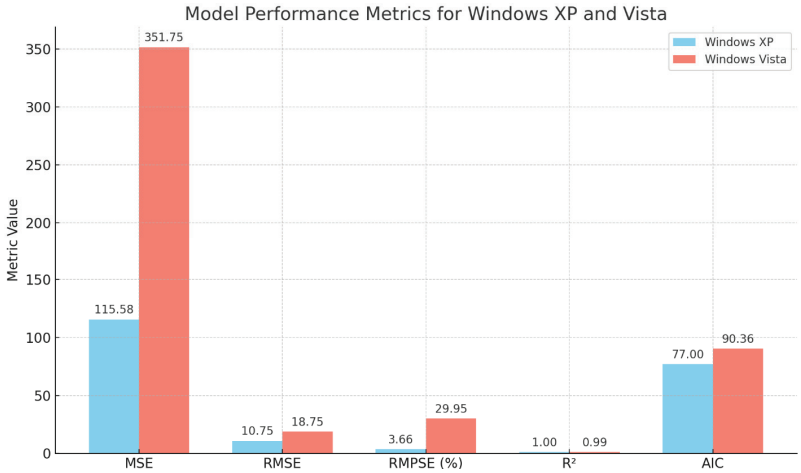

Figure 2 Evaluation metrics for Windows XP and Windows Vista vulnerability discovery models.

lower for the XP model, signifying a more optimal balance between goodness of fit and model complexity. Overall, the metrics confirm that the Windows XP model outperforms the Vista model across all evaluated dimensions.

The model estimates a significantly larger total vulnerability count for Windows XP than the number observed. This reflects a long tail of latent vulnerabilities that likely remained undiscovered, especially after XP reached end-of-life and researcher attention shifted elsewhere. Such long-tail behaviour is typical in legacy systems and highlights the residual security risk that persists even when public reporting declines.

Furthermore, the observed asymmetry in cross-discovery influence, where discoveries in XP have a greater impact on Vista than the other way around, has a plausible security explanation. XP had a larger installed base, longer exposure, and a more diverse third-party ecosystem, which made vulnerabilities originating from XP more transferable to Vista. In contrast, Vista introduced several architectural changes that reduced the backward transfer of vulnerabilities to XP. This asymmetry fits with known differences in system design and ecosystem maturity.

6 Cross-Discovery Analysis

To capture the interplay between successive software releases in vulnerability discovery, the author models cross-discovery effects between Windows XP

Table 5 Direct and cross-discovered vulnerabilities for Windows XP (R1) and Vista (R2)

Year	R1 Discovered	R2 Discovered	R1 from R2	R2 from R1
2012	2.16	25.21	65.53	24.72
2013	1.83	13.87	35.91	24.50
2014	1.55	6.60	17.02	24.26
2015	1.32	2.92	7.51	24.02

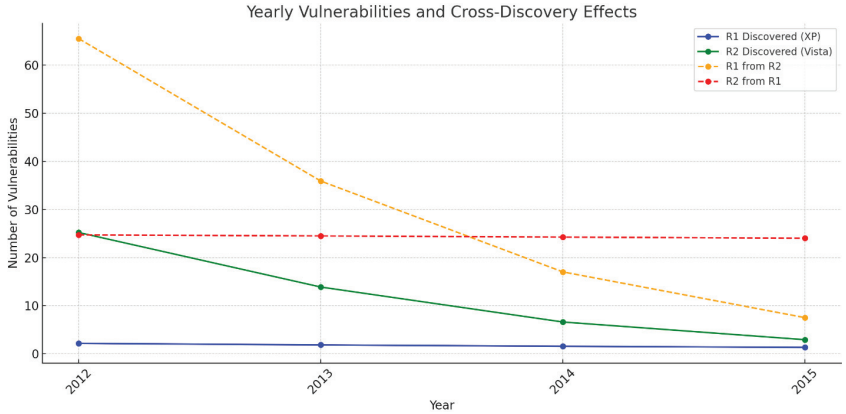

Figure 3 Yearly discovered and cross-discovered vulnerabilities in Windows XP and Vista (2012–2015).

(Release 1) and Windows Vista (Release 2). This phenomenon suggests that vulnerabilities found in one version may trigger or relate to vulnerabilities in another version, due to shared architecture, code reuse, or patch analysis. The authors evaluate the model during the overlapping period $t = 12$ to $t = 15$, corresponding to the years 2012–2015. The table below presents the annual number of vulnerabilities discovered in each release, along with the estimated number of discoveries attributable to the influence of the other release:

The influence of Vista (R2) on XP (R1), captured by α_{12}, starts high in 2012 (65.53) and declines over time, aligning with a positive decay rate γ_{12}. The direct discoveries in both releases are relatively low in these years, highlighting the increasing proportion of discoveries driven by inter-release dynamics.

The plot illustrates the yearly discoveries of vulnerabilities in both Windows XP (Release 1) and Windows Vista (Release 2), along with their mutual influence (cross-discovery effects) from 2012 to 2015:

Blue Line (R1 Discovered): Vulnerabilities directly found in Windows XP. The count is low and slowly decreasing over time. Green Line (R2

Discovered): Vulnerabilities in Windows Vista. These decline more sharply after 2012. Orange Dashed Line (R1 from R2): Vulnerabilities in XP that are influenced by discoveries in Vista. This starts high and declines, showing decreasing cross-influence from Vista over time. Red Dashed Line (R2 from R1): Vulnerabilities in Vista influenced by XP.

7 Conclusion

In this study, author proposed a time-dependent mathematical model to analyze how software vulnerabilities are discovered across multiple releases, using Windows XP and Windows Vista as case examples. Unlike traditional multi-release VDMs that assume constant cross-discovery rates, our model introduces a dynamic interaction component that captures how the discovery process evolves over time. By fitting the model to real historical vulnerability data, the author showed that it accurately reflects the long-term discovery patterns of both releases, with particularly strong performance for Windows XP due to its more stable reporting trends.

The results also reveal clear asymmetry in cross-version influence: discoveries in XP contributed more strongly to identifying vulnerabilities in Vista than the reverse. The author discusses plausible architectural and ecosystem-based reasons for this asymmetry and highlights how the model implies a long tail of latent vulnerabilities, especially in legacy systems.

Although classical baseline methods could not be reliably fitted to the sparse XP–Vista dataset, our model offers meaningful insights into unfolding vulnerability dynamics and provides a general framework for analyzing multi-release systems. As software ecosystems evolve toward rapid CI/CD pipelines, the time-dependent structure enables easy adaptation to shorter release cycles. Future work can extend this approach to richer datasets, incorporate machine learning for dynamic parameter updates, and analyze multi-product influence networks to support proactive and data-driven software security planning.

8 Future Scope

The proposed model presents several promising directions for future research. One important extension is to apply the model to a wider variety of software ecosystems, including mobile operating systems, cloud infrastructures, and large-scale open-source projects. Such validation would test the robustness and generalizability of the model beyond the Windows family. Another useful

direction is the integration of additional factors that influence vulnerability discovery, such as patch release timing, exploit availability, community reporting behavior, and internal development practices. Incorporating these factors could lead to richer and more realistic modeling outcomes. Future work may also extend the framework to explicitly capture zero-day vulnerabilities and post-patch vulnerability emergence, making the model more suitable for real-time security analysis. Techniques from machine learning and Bayesian updating can be combined with the current model to enable dynamic parameter estimation as new data becomes available. Additionally, the concept of cross-influence can be expanded to construct multi-version or multi-product vulnerability propagation networks, allowing researchers to analyze entire software families rather than just pairwise interactions. These enhancements will help refine the predictive capability of the model and support its use in proactive security planning and automated vulnerability management.

9 Applications of the Proposed Work

The proposed model offers several practical applications in software security and maintenance. One major application is in vulnerability forecasting, where security teams can use model predictions to anticipate future discovery rates and prepare mitigation strategies in advance. This is especially valuable in environments where multiple versions of the same software are deployed simultaneously. The model also supports improved patch planning and prioritization, as insights into cross-version influence help organizations determine which releases require urgent attention.

Another important application lies in risk assessment. By quantifying how vulnerabilities propagate across releases, organizations can evaluate the security posture of legacy versions and make informed decisions about system upgrades or retirement. The model can also contribute to secure software development lifecycles by revealing stages in which vulnerability discovery accelerates, enabling developers to strengthen testing and code review efforts. Furthermore, vendors can use the model to coordinate updates across related software products, reducing the probability of shared vulnerabilities being exploited. Policymakers and large enterprises may leverage the model to guide investment in security resources and migration planning. Overall, the model serves as a valuable analytical tool for enhancing software reliability, strengthening security strategies, and improving long-term maintenance decisions.

10 Limitations/Methodology Discussion

In traditional multi-release VDMs, cross-discovery effects are typically assumed to be constant over time. The author tried fitting these constant coefficient models to the Windows XP and Vista datasets, but the sparse observations and irregular reporting periods made the classical approach statistically unstable and, in some cases, non-identifiable. Due to this instability, producing meaningful numerical comparisons was not feasible. Instead, it is important to highlight that the proposed time-dependent formulation still includes the traditional framework. When the influence term remains fixed, the new formulation reduces to the classic multi-release VDM. Although a direct empirical comparison cannot be made with this dataset, the conceptual connection between the two methods remains valid.

A predictive evaluation, where the model is trained on an early part of the dataset and then used to predict future discoveries, would ideally be included. However, the XP and Vista dataset is too limited and unevenly distributed to support stable predictive analysis. Once the data is divided, for example, by training only on observations up to 2012, parameter estimates become very unstable because of the small sample size and long gaps in reporting. As a result, the forecasts that follow do not reflect meaningful model behavior. For this reason, predictive validation could not be incorporated. Future research using richer and more continuous datasets, such as those available for modern browsers, Android, or the Linux kernel, may enable reliable forecasting experiments.

Although the dataset comes from Windows XP and Vista, the core design of the model is general and can be applied to modern rapid release CI and CD environments. In fast-moving development ecosystems, vulnerability discovery periods are shorter, patch cycles are more frequent, and many small releases happen simultaneously. The time-dependent cross influence parameter used by the author is adaptable enough to reflect these conditions by recalibrating the influence function to match shorter software lifecycle intervals. In this way, the current study serves as a proof of concept that can be tailored to high-velocity development pipelines when contemporary datasets become available.

References

[1] Krsul, I. V. (1998). Software vulnerability analysis. West Lafayette, IN: Purdue University.

[2] Massacci, F., and Nguyen, V. H. (2014). An empirical methodology to evaluate vulnerability discovery models. IEEE Transactions on Software Engineering, 40(12), 1147–1162.
[3] Rescorla, E. (2005). Is finding security holes a good idea?. IEEE Security & Privacy, 3(1), 14–19.
[4] Needham, R. M. (2002). Security and open source. Open Source Software Economics.
[5] Alhazmi, O. H., and Malaiya, Y. K. (2005, November). Modeling the vulnerability discovery process. In 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05) (pp. 10-pp). IEEE.
[6] Woo, S. W., Joh, H., Alhazmi, O. H., and Malaiya, Y. K. (2011). Modeling vulnerability discovery process in Apache and IIS HTTP servers. Computers & Security, 30(1), 50–62.
[7] Alhazmi, O. H., Malaiya, Y. K., and Ray, I. (2007). Measuring, analyzing and predicting security vulnerabilities in software systems. Computers & Security, 26(3), 219–228.
[8] Joh, H., Kim, J., and Malaiya, Y. K. (2008, November). Vulnerability discovery modeling using Weibull distribution. In 2008 19th International Symposium on Software Reliability Engineering (ISSRE) (pp. 299–300). IEEE.
[9] Younis, A., Joh, H., and Malaiya, Y. (2011). Modeling learningless vulnerability discovery using a folded distribution. In Proc. of SAM (Vol. 11, pp. 617–623).
[10] Joh, H., and Malaiya, Y. K. (2014). Modeling skewness in vulnerability discovery. Quality and Reliability Engineering International, 30(8), 1445–1459.
[11] Kapur, PK, Sachdeva, N, Khatri, SK. (2015). Vulnerability Discovery Modeling. International Conference on Quality, Reliability, Infocom Technology and Industrial Technology Management 34–54.
[12] Anand, A., and Bhatt, N. (2016). Vulnerability Discovery Modeling and Weighted Criteria Based Ranking. Journal of the Indian Society for Probability and Statistics, 1–10.
[13] Bhatt, N., Anand, A., Yadavalli, V.S.S. and Kumar, V. (2017) 'Modeling and Characterizing Software Vulnerabilities', International Journal of Mathematical, Engineering and Management Sciences, Vol. 2, No. 4, pp. 288–299
[14] Bhatt, N., Anand, A., and Aggrawal, D. (2020). Improving system reliability by optimal allocation of resources for discovering software

vulnerabilities. International Journal of Quality & Reliability Management, 37(6/7), 1113–1124.

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

[16] Anand, A., Bhatt, N., and Alhazmi, O. H. (2021). Modeling software vulnerability discovery process inculcating the impact of reporters. Information Systems Frontiers, 23(3), 709–722.

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

[18] Anand, A., Aggrawal, D., and Alhazmi, O. H. (2025). Multi-Phase Modeling for Vulnerability Detection & Patch Management: An Analysis Using Numerical Methods. *Computers, Materials & Continua*, *84*(1).

[19] Divya, Anand, A., Bhatt, N., and Johri, P. (2025). 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*(02), 2450031.

[20] Kim, J., Malaiya, Y. K., and Ray, I. (2007, November). Vulnerability discovery in multi-version software systems. In High Assurance Systems Engineering Symposium, 2007. HASE'07. 10th IEEE (pp. 141–148). IEEE.

[21] Anand, A., Das, S., Aggrawal, D., and Klochkov, Y. (2017). Vulnerability Discovery Modelling for Software with Multi-versions. In Advances in Reliability and System Engineering (pp. 255–265). Springer International Publishing.

[22] Kapur PK, Pham H, Gupta A, Jha PC. (2011) Software Reliability assessment with OR application. Springer: Berlin.

[23] Windows Xp (2018). Vulnerability Statistics. http://www.cvedetails.com/product/739/Microsoft-Windows-Xp.html?vendor_id=26.

[24] Windows Vista (2018). Vulnerability Statistics. https://www.cvedetails.com/product/9591/Microsoft-Windows-Vista.html?vendor_id=26.

Biographies

Jyotish N. P. Singh is an Assistant Professor of Decision Sciences, an academic with expertise in Operational Research, Mathematics, and Software Reliability, known for his research, publications, and mentorship, associated with NICMAR Pune and Delhi University. He's an author on software reliability and mathematical modeling, a public speaker, and passionate about India's contributions to math, while also dabbling in poetry.

Navneet Bhatt is an Assistant Professor at SVKM's NMIMS Anil Surendra Modi School of Commerce. He holds a PhD and MPhil in Operational Research and specializes in mathematical modeling, quantitative techniques, machine learning, and software vulnerability analytics, with several research publications and academic contributions.

Adarsh Anand did his doctorate in the area of Innovation Diffusion Modeling in Marketing and Software Reliability Assessment. Presently he is working as an Associate Professor in the Department of Operational Research, University of Delhi (INDIA). He has been conferred with Young Promising Researcher in the field of Technology Management and Software Reliability by Society for Reliability Engineering, Quality and Operations Management (SREQOM) in 2012. He is a lifetime member of the Society for Reliability Engineering, Quality and Operations Management (SREQOM). He is also on the editorial board of International Journal of System Assurance and Engineering management (Springer). He has Guest edited several Special Issues for Journals of international repute. He has edited two books namely: "System Reliability Management (Solutions and Technologies)" and "Recent Advancements in Software Reliability Assurance" under the banner of Taylor and Francis (CRC – Press). He has publications in journals of national and international repute. His research interest includes modeling innovation adoption and successive generations in marketing, software reliability growth modelling and social media analysis.

Divya received a bachelor's degree in Mathematics from University of Delhi in 2020 and a master's degree in operational research from University of Delhi in 2022, and she is currently pursuing the philosophy of doctorate degree in operational research from University of Delhi.