CITY, UNIVERSITY OF LONDON

DOCTORAL THESIS

---

# Analyzing CVE Lifecycle and Relationships with Exploits, Patches, CWEs, and CPEs: A Focus on Microsoft Vulnerabilities

---

*Author:*
Eid ALBADDAH

*Supervisor:*
Dr. Ilir GASHI, Dr. Jacob HOWE

*A thesis submitted in fulfillment of the requirements for the degree of Doctor of Cybersecurity*

*in the*

Information Security
Department of Computer Science

August 2, 2025

# Declaration of Authorship

I, Eid ALBADDAH, declare that this thesis titled, “Analyzing CVE Lifecycle and Relationships with Exploits, Patches, CWEs, and CPEs: A Focus on Microsoft Vulnerabilities” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

*"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."*

Dave Barry

CITY, UNIVERSITY OF LONDON

# *Abstract*

CS
Department of Computer Science

Doctor of Cybersecurity

**Analyzing CVE Lifecycle and Relationships with Exploits, Patches, CWEs, and CPEs: A Focus on Microsoft Vulnerabilities**

by Eid ALBADDAH

In today's interconnected world, the security of information systems is of paramount importance. The continuous discovery of vulnerabilities, their exploitation, and subsequent patching significantly influence the security landscape. This thesis aims to analyze security vulnerabilities from all aspects as well as the lifecycle of Common Vulnerabilities and Exposures (CVEs) and their relationships with exploits, weaknesses (CWEs), and affected products (CPEs), with a particular focus on CVEs related to Microsoft. By leveraging a comprehensive dataset, this research will extend the work presented in Stefan Frei's 2009 dissertation [46], "Security Econometrics: The Dynamics of (In)Security," to understand the dynamics between the discovery, exploitation, and patching of vulnerabilities. Our study utilizes data from multiple sources, including CVE V5, ExploitDB, and Microsoft's MSRC API, to examine trends and patterns in the race between exploits and patches, providing valuable insights into the security ecosystem.. . .

# *Acknowledgements*

Special thanks to my supervisors Dr. Ilir GASHI, Dr. Jacob HOWE for their tremendous support during my journey…

# Contents

# List of Figures

# LIST OF TABLES

# List of Abbreviations

| | |
|---|---|
| **CVE** | Common Vulnerability Enumeration |
| **CWE** | Common Weakness Enumeration |
| **CPE** | Common Product Enumeration |
| **MSRC** | Microsoft Security Response Center |
| **API** | Application Programming Interface |
| **CVSS** | Common Vulnerability Scoring System |
| **CVRF** | Common Vulnerability Reporting Framework |
| **OSVDB** | Open Source Vulnerability Database |
| **CERT** | Coordination Center for Emergency Response Teams |
| **CERT-CC** | CERT Coordination Center (Carnegie Mellon University) |
| **US-CERT** | United States Computer Emergency Readiness Team |
| **(ISC)$^2$** | International Information Systems Security Certification Consortium |
| **NIST** | National Institute of Standards and Technology |
| **DDoS** | Distributed Denial-of-Service |
| **XSS** | Cross-Site Scripting |
| **SQLi** | SQL Injection |
| **CSRF** | Cross-Site Request Forgery |
| **IoT** | Internet of Things |
| **PII** | Personally Identifiable Information |
| **(ISC)$^2$ CIS Controls** | The Center for Internet Security Critical Security Controls |

*For/Dedicated to/To my…*

# CHAPTER 1

# INTRODUCTION

## 1.1 Background

In today's interconnected world, the security of information systems is of paramount importance. The continuous discovery of vulnerabilities, their exploitation, and subsequent patching significantly influence the security landscape. This thesis aims to use open-source intelligence to help improve evidence-based security management and decision making. In the work done to date we have analysed Common Vulnerabilities and Exposures (CVEs) data as well as their lifecycle and relationships with exploits, weaknesses (CWEs), and affected products (CPEs), and patches. The patch information, so far, has focused on Microsoft products.

CVEs are publicly disclosed cybersecurity vulnerabilities that are assigned an identifier for tracking purposes [108]. By leveraging comprehensive public datasets, this research will extend the work presented in Stefan Frei's 2009 dissertation [46], "Security Econometrics: The Dynamics of (In)Security," to understand the dynamics between the discovery, exploitation, and patching of vulnerabilities.

Our study utilizes data from multiple public sources, including:

- CVE V5 database maintained by MITRE [108]
- ExploitDB, a comprehensive archive of public exploits [112]
- Microsoft's MSRC API, providing patching information [104]

These sources allow us to examine trends and patterns in the race between exploits and patches, providing valuable insights into the security ecosystem. The focus on Microsoft-related vulnerabilities is influenced not only by the company's significant market share but also by the availability of comprehensive patching data and its clear links with CVEs. This allows for a more detailed analysis of the full vulnerability lifecycle.

While our patching analysis centres on Microsoft, our examination of vulnerabilities and exploits extends to non-Microsoft products as well, providing a broader perspective on the security landscape. In the next stage of the research, we will be looking at adding more data-sources for patches of other vendors.

## 1.2 Objectives

As stated previously, the main objective of the thesis is to use open-source data to improve security decision making. This work builds upon several foundational studies in vulnerability management, particularly Frei's seminal work [46] on the dynamics of (in)security that established frameworks for understanding vulnerability lifecycles. Despite significant advances in the field, critical gaps remain in comprehensively analysing the relationships between CVEs, exploits, patches, and weakness categories, which this research aims to address. The sub-objectives, which help in achieving this higher level objective, are:

1. **Analyse CVEs with their corresponding weaknesses (CWEs), affected products (CPEs), exploits, and patching information.** This extends previous work by Turtiainen and Costin [153] on automated CWE classification and addresses data quality challenges identified by Anwar et al. [13] in their assessment of the National Vulnerability Database.

2. **Gain a comprehensive understanding of the CVE ecosystem by identifying patterns and trends.**
   While Rajasooriya et al. [123] developed statistical models using Markov chain theory to estimate vulnerability exploitation probabilities, our work aims to provide a more integrated analysis that accounts for the relationships between different vulnerability attributes.

3. **Analyse the distribution and timing of key events in the vulnerability lifecycle from 2016 to 2024, with a focus on Microsoft-related vulnerabilities.**
   This builds upon Shahzad et al.'s [134], [135] research on Microsoft's structured patch release cycles and Dissanayake's [37] examination of temporal patterns in Microsoft security responses, providing updated insights for contemporary security environments.

4. **Identify trends and patterns in the lifecycle stages, particularly focusing on discovery, exploit availability, and patch availability.**
   Ruohonen [130] documented significant variations in time between discovery and public disclosure, with our preliminary analysis supporting these findings by revealing a median disclosure time of 41 days but a considerably higher mean of 141.9 days, indicating substantial variability in disclosure practices.

5. **Investigate outliers and significant deviations in the lifecycle distribution to understand their impact on security.**
   This addresses the challenges posed by censored data in vulnerability lifecycle analysis identified by Brilhante et al. [27], who implemented heavy-tailed distributions for modelling vulnerability timelines.

6. **Provide actionable insights for improving vulnerability management practices based on empirical data.**
   Recent work by Woo et al. [161] revealed that many security patches require subsequent modifications to achieve their protection goals, whilst Xiong et al. [162] demonstrated how the quality of vulnerability information disclosure directly impacts vendors' patch development capabilities. Our research aims to address these challenges through empirical analysis.

7. **Develop predictive models using machine learning techniques to gain deeper insights into vulnerability lifecycles and exploit likelihood.**
   This extends Jacobs et al.'s [67], [68] Exploit Prediction Scoring System (EPSS) and Suciu et al.'s [143] Expected Exploitability (EE) metric by incorporating a more comprehensive set of variables derived from our integrated dataset.

8. **Explore the potential of Time To Patch modeling for common types of vulnerabilities.**
   Building upon recent advances in machine learning for vulnerability analysis, this objective aims to address the gap between vulnerability awareness and effective patch releasing in enterprise environments.

So far we have made progress with sub-objectives 1-6. In the next stage of the PhD, we plan to work more on sub-objectives 7-8 as well as revisit the other sub-objectives above.

## 1.3 Methodology

Our analysis leverages a comprehensive OSINT dataset of vulnerabilities (CVEs), exploits, and patches spanning multiple years. Data was collected from authoritative sources including the CVE V5 [108] database maintained by MITRE, ExploitDB [112], and Microsoft's Security Response Center (MSRC) API [104], ensuring reliability and representativeness of the security landscape. For CVE and exploit data, we extracted information programmatically using automated ETL (Extract, Transform, Load) pipelines from GitHub repositories, while patch information was retrieved through API calls to the Microsoft MSRC database.

We processed this data through a rigorous pipeline that included data cleaning, normalization, and structured integration into a MySQL database. For temporal analysis, we consistently used the earliest available dates for key lifecycle events (CVE reservation, exploit publication, and patch release) to ensure accuracy in measuring vulnerability lifecycles. Special processing workflows were developed for specific analyses, such as product family distribution and CWE classification, addressing challenges with data completeness and consistency.

For the analysis of vulnerability distribution across Microsoft product families, we utilized MSRC product information rather than CVE-V5 CPE data after discovering that 30.6% of patched CVEs lacked CPE information. This methodological decision ensured comprehensive coverage while accurately handling many-to-many relationships between vulnerabilities and products. Similarly, for CWE distribution analysis, we merged MSRC data with the CVE-V5 database, focusing on the 5,374 CVEs (53.8%) with identifiable CWE classifications while acknowledging the limitations of this coverage gap.

The NVD [111] dataset was incorporated to supplement information on Common Weakness Enumeration (CWE), comprehensive CVSS scores, and CPE (product) data, linking vulnerabilities to their underlying software weaknesses and affected products. This integration enabled deeper analysis of vulnerability patterns and trends, including the examination of CWE pairs and triplets that reveal complex relationships between different weakness types.

Statistical analysis was employed to examine the distribution of lifecycle events, calculating mean, median, and mode differences between these events along with dispersion measures to quantify variability. This multi-indicator approach provided a more nuanced understanding of temporal relationships than previous studies that often relied solely on averages. Our dataset comprises non-duplicated vulnerabilities and exploit records for multiple vendors, with a specific focus on Microsoft-related patches due to their comprehensive documentation and clear linkage with CVEs.

For the complete lifecycle analysis, we focused on 415 Microsoft-related CVEs with documentation for all three key events (CVE reservation, exploit publication, and patch availability). This subset was further categorized into Microsoft-specific products (391 CVEs, 94%) and third-party software operating on Microsoft platforms (24 CVEs, 6%), enabling detailed comparison of lifecycle patterns between these categories.

The methodology ensures that our findings are based on empirical data spanning multiple years, providing insights into vulnerability management practices and their evolution over time. The collected data is updated through April 2024, ensuring contemporary relevance of our analysis.

## 1.4 Contributions

This study makes several key contributions to the field of cybersecurity:

1. A detailed analysis of the vulnerability lifecycle over a nine-year period (2016-2024), revealing that exploits are typically published approximately 168 days after CVE creation (mean value) with a median of 141 days, while patches typically appear 7 days before exploits (median of Patch - Exploit = -7 days).

2. Identification of significant temporal patterns in patch deployment, with our findings showing that 93.7% of analyzed Microsoft-related CVEs were patched before being exploited, demonstrating effective vulnerability disclosure mechanisms despite the negative median gap between exploit and patch dates.

3. Evidence of substantial discrepancies between vendor-reported exploit data and third-party databases, with our analysis revealing that out of 124 CVEs reported as exploited by Microsoft, only 29 (23.4%) were documented in the Exploit Database, highlighting critical gaps in vulnerability intelligence sharing and reporting.

4. Comprehensive examination of vulnerability weaknesses (CWEs), demonstrating that Cross-site Scripting (CWE-79) remains the most prevalent weakness at 15.35% of all vulnerabilities, followed by Memory Buffer issues (CWE-119) at 6.84% and SQL Injection (CWE-89) at 6.52%, with persistence of these issues despite decades of awareness.

5. Discovery of significant changes in vulnerability distributions during the pandemic period (2020-2021), with a 78% increase in CVEs attributed to rapid digital transformation and increased focus on security research, exceeding the consistent 10% annual growth predicted by industry reports.

6. Documentation of the dramatic evolution in exploitation patterns over time, with a shift from predominantly pre-CVE exploits in early years (1999-2004, over 80%) to predominantly post-CVE exploits in recent years (2015-2024, over 80%), reflecting significant improvements in coordinated vulnerability disclosure practices.

7. Analysis of exploitation patterns revealing counterintuitive severity-exploitation relationships, with critical vulnerabilities showing longer exploitation timelines (median 192 days) than medium-severity ones (median 138 days), challenging conventional prioritization approaches based solely on severity ratings.

8. Identification of dramatic disparities in patching times between exploited and non-exploited vulnerabilities, with exploited critical vulnerabilities taking 244% longer to patch than non-exploited ones (186 vs. 54 days), suggesting that exploited vulnerabilities may possess characteristics that significantly complicate remediation.

9. Revelation of significant differences between Microsoft-specific products and third-party components in lifecycle patterns, with only 0.26% of Microsoft-specific products (1 out of 391) exhibiting extreme lifecycle delays exceeding 1,000 days, compared to 37.5% of third-party applications (9 out of 24).

10. Development of a novel framework for analyzing CVEs with multiple exploits, showing that 61% of vulnerabilities with 10-13 exploits exhibited consistent characteristics in exploitation methods, with this consistency decreasing as the number of exploits per CVE decreases.

11. Discovery of distinctive product-specific vulnerability concentrations, with Microsoft Office leading with 6,342 patched vulnerabilities, significantly ahead of Windows (4,869) and .NET (4,867), challenging assumptions that operating systems represent the most vulnerable components.

12. Identification of Microsoft's distinctive security profile compared to industry norms, with memory-related weaknesses accounting for 58.1% of Microsoft's top patched vulnerabilities (compared to 32.1% in the broader vulnerability landscape), while web vulnerabilities constitute only 7.4% (versus 38.1% industry-wide).

13. Creation of a comprehensive dataset and methodology that can serve as a reference for future research in vulnerability management, enabling more informed decision-making in security operations.

14. Potential applications for machine learning models to predict exploit likelihood and optimize vulnerability management processes, based on the temporal patterns identified in our analysis.

## 1.5 Document Guide

Throughout this thesis, colored boxes are used to highlight specific types of information:

- **Example Key Insights Box**

  Blue boxes contain key insights, summaries of findings, and important concepts. These boxes are used to highlight the most significant results from analyses and their implications.

- **Example Information Box**

  Yellow boxes with red borders provide important contextual information, examples from literature, or critical considerations that deserve special attention.

- (Note) indicated below provides brief clarifications or additional points of interest related to the main text.
- **Note:** Example note

These visual elements are designed to help readers quickly identify different types of information and navigate the content more effectively.

## 1.6 Report Structure

The remainder of this thesis is organized as follows:

**Chapter 2: Literature Review** provides a comprehensive overview of existing research on vulnerability lifecycles, exploit prediction, and patch management.

**Chapter 3: Methodology** details the data sources, collection methods, and analytical techniques employed in this study.

**Chapter 4: Vulnerability, Exploit, and Patch Analysis** presents a detailed examination of CVEs, their associated exploits, and patching trends.

**Chapter 5: Lifecycle Analysis** offers an in-depth analysis of vulnerability lifecycles, focusing on the timing and sequence of key events of interest (i.e. vulnerability discovery, exploitation and patching).

**Chapter 6: Future Work** Outlines potential areas for further research and development, including advanced analytics and machine learning applications.

By building upon existing research and providing a comprehensive understanding of the vulnerability lifecycle, this study aims to enhance the effectiveness of vulnerability management practices and contribute to the broader field of cybersecurity research.

# CHAPTER 2

# Literature Review

## 2.1 Introduction

The landscape of cybersecurity vulnerabilities has evolved dramatically in recent years, driven by increasing system complexity, rapid software development cycles, and sophisticated cyber threats. Understanding this evolution requires leveraging Open-Source Intelligence (OSINT), a methodological framework that transforms publicly available information into actionable intelligence through systematic collection, processing, and analysis [117]. In cybersecurity research, OSINT encompasses a wide range of information sources, from technical databases to social media discussions, providing researchers and security professionals with comprehensive insights without requiring privileged access or specialized permissions [109].

Among the various applications of OSINT in cybersecurity, vulnerability intelligence represents one of its most structured and standardized domains [87]. This standardization manifests in well-defined relationships between Common Vulnerabilities and Exposures (CVEs), their associated exploits, patches, and classifications through Common Weakness Enumeration (CWE) and Common Platform Enumeration (CPE). The temporal aspects of these relationships – specifically when vulnerabilities are reported, when patches become available, and when exploits emerge – provide crucial insights into the effectiveness of current security practices and the dynamics of the threat landscape [130].

The systematic collection and analysis of vulnerability-related OSINT has transformed security research by providing rich, publicly accessible datasets that enable deeper understanding of vulnerability lifecycles [131]. These sources include the National Vulnerability Database (NVD), ExploitDB, vendor security advisories, and increasingly, social media platforms and code repositories that offer real-time insights into emerging threats and exploitation patterns [117]. While this wealth of information presents unprecedented opportunities for comprehensive analysis, it also introduces significant challenges in data integration, quality assessment, and temporal alignment [51], [109].

Recent research demonstrates that understanding vulnerability lifecycles requires consideration of multiple interconnected factors. Frei's seminal work [46] established the foundational framework for analyzing vulnerability lifecycles, while more recent studies have expanded this understanding through the integration of machine learning approaches and real-time threat intelligence [4]. This evolution in research methodology reflects the growing recognition that effective vulnerability management requires dynamic, data-driven approaches that can adapt to rapidly changing threat landscapes [27].

Microsoft vulnerabilities present a particularly significant area of study within the OSINT-driven vulnerability research landscape due to the company's substantial market presence and the extensive documentation available about their security response processes [37]. The structured nature of Microsoft's security update cycles, combined with comprehensive vulnerability documentation, provides unique opportunities for analyzing the relationships between vulnerability characteristics, exploitation patterns, and patch deployment strategies.

Understanding vulnerability management requires familiarity with several interconnected frameworks and concepts that form the foundation of modern security analysis. The Common Vulnerabilities and Exposures (CVE) system, maintained by MITRE Corporation, provides a standardized method for identifying and cataloging known cybersecurity vulnerabilities [111]. Each CVE entry represents a specific vulnerability with a unique identifier, enabling consistent tracking and reference across different security tools and databases. This standardization has proven crucial for coordinated vulnerability management across the industry [100].

The Common Weakness Enumeration (CWE) framework offers a comprehensive classification of software security weaknesses [94]. Unlike CVEs, which identify specific vulnerabilities, CWEs describe types of weaknesses that can lead to vulnerabilities. This classification system enables understanding of common vulnerability patterns and supports development of more secure software by identifying recurring security issues [95].

Common Platform Enumeration (CPE) provides a structured naming scheme for identifying and categorizing IT systems, software packages, and platforms [29]. This standardization enables precise tracking of which systems are affected by specific vulnerabilities, facilitating more effective vulnerability management across complex IT environments. The CPE framework's hierarchical structure allows for detailed specification of affected systems while maintaining flexibility for different deployment scenarios.

The exploitation of vulnerabilities represents the practical manifestation of these security weaknesses in real-world scenarios [4]. An exploit demonstrates how a vulnerability can be leveraged to compromise system security, ranging from proof-of-concept demonstrations to fully weaponized attack tools. Understanding exploitation patterns through OSINT sources has become crucial for assessing real-world vulnerability impacts and prioritizing remediation efforts [51].

Patches represent the remediation component of vulnerability management, providing fixes for identified security issues. The patching process involves complex decisions about timing, testing, and deployment strategies, particularly in enterprise environments where system stability must be balanced against security needs [37]. Microsoft's structured approach to patch management through its monthly "Patch Tuesday" releases has become a de facto standard in the industry, though it also creates predictable patterns that attackers may exploit [27].

Other major vendors employ diverse patching strategies with varying levels of structure and predictability. Oracle follows a quarterly Critical Patch Update (CPU) schedule, releasing security fixes every January, April, July, and October, while Apple adopts a more irregular patching model driven by vulnerability severity rather than fixed schedules [135]. Google has implemented a hybrid approach for Chrome, combining scheduled releases with expedited patches for critical vulnerabilities [151]. In contrast, many open-source projects like the Linux kernel employ a distributed model relying on community developers to identify, fix, and validate patches through transparent processes [86]. Researchers have analyzed these varied approaches using data from multiple sources. Li et al. [86] utilized the National Vulnerability Database alongside Git repositories to examine 4,013 Linux kernel security patches, while Thomas et al. [151] leveraged Chrome's release data and vulnerability reports to analyze Google's vulnerability management practices. Ozment and Schechter [114] obtained OpenBSD patch data directly from project's security advisories and source repositories, demonstrating diverse methodologies for acquiring vendor-specific patch information. A comparative analysis by Shahzad et al. [134] revealed significant disparities in patch development times across vendors, with Apple and Mozilla demonstrating more rapid response capabilities than Oracle for vulnerabilities of similar severity. Understanding these cross-vendor patterns provides crucial context for developing comprehensive vulnerability management strategies applicable across heterogeneous software environments.

This review synthesizes current research across several key areas within the OSINT-driven vulnerability research domain: vulnerability lifecycle analysis, exploitation patterns, patch management strategies, and predictive modeling approaches. Particular attention is paid to the integration of multiple data sources and the application of modern analytical techniques, including machine learning and natural language processing, in vulnerability assessment and management. The review also examines how recent advances in automated vulnerability analysis and risk assessment have transformed traditional approaches to security management.

The following sections examine these themes in detail, beginning with key concepts in vulnerability analysis and progressing through empirical studies, methodological approaches, and current gaps in research. This structure enables comprehensive understanding of both the theoretical foundations and practical applications of vulnerability management, while highlighting areas where further research is needed.

## 2.2 Definitions and Types of Vulnerabilities

In the realm of cybersecurity, a vulnerability refers to a weakness or flaw in a software system, network, or process that can be exploited by an attacker to gain unauthorized access, cause damage, or steal sensitive data [100]. The academic literature presents several established frameworks for classifying vulnerabilities, each offering different perspectives on how to categorize and understand these security weaknesses. While classification systems represent just one aspect of vulnerability management, they provide essential foundations for understanding how security weaknesses manifest and evolve in modern systems.

### 2.2.1 Historical Development of Vulnerability Classification

The evolution of vulnerability classification frameworks reflects our growing understanding of security weaknesses. The foundational work in vulnerability classification emerged with Landwehr et al. [84], who developed a comprehensive taxonomy that categorized vulnerabilities along three critical dimensions: genesis, time of introduction, and location. Building upon this taxonomic foundation, Bishop [23] advanced the field by creating a specialized classification system focused specifically on Unix systems. However, subsequent critical analysis by Bishop and Bailey [24] revealed inherent limitations in these early taxonomic approaches, particularly highlighting their inconsistency when applied across different levels of abstraction. As the field matured, Krsul [79] introduced a novel perspective by examining vulnerabilities through the lens of environmental assumptions, though the practical implementation of this theoretical framework proved challenging. That same year, Du and Mathur [38] proposed a more pragmatic classification approach, categorizing software errors based on three key aspects: their cause, impact, and fix type, which provided practitioners with a more actionable framework. Piessens [120] contributed to this evolution by proposing a systematic taxonomy focused on Internet software vulnerabilities, while Pothamsetty and Akyol [121] developed a specialized classification system for network protocols. A significant advancement came with Howard et al. [61], who introduced the "19 deadly sins" approach, offering one of the most comprehensive classification systems of its time. Weber et al. [159] further refined the theoretical understanding of vulnerability classifications by arguing that their practical utility should take precedence over strict scientific correctness, acknowledging the inherent challenges in developing purely scientific taxonomies.

### 2.2.2 Recent Classification Frameworks

Meunier's work [102] provided critical insights into classification challenges, emphasizing that vulnerabilities exist as conceptual entities at multiple abstraction levels simultaneously. This understanding helped explain why early classification attempts often failed to meet scientific criteria of objectivity, determinism, repeatability, and specificity.

Frei's influential research [46] introduced a temporal classification framework based on vulnerability discovery and disclosure status, identifying three main categories:

- Publicly disclosed vulnerabilities known to vendors and security communities
- Privately reported vulnerabilities disclosed through coordinated processes
- Undisclosed vulnerabilities potentially known to attackers but not reported to vendors

The Common Weakness Enumeration (CWE) system [94] further refined these approaches, offering more granular classification based on underlying weakness types. The National Institute of Standards and Technology (NIST) [99] and Da Silva et al. [140], in *A Catalog of Security Architecture Weaknesses*, provide another comprehensive classification framework that categorizes vulnerabilities based on their technical nature. This framework identifies several fundamental types of security weaknesses:

- **Design Vulnerabilities**: Fundamental flaws in system architecture or security controls
- **Implementation Vulnerabilities**: Weaknesses introduced during development
- **Operational Vulnerabilities**: Weaknesses arising from improper deployment

Recent advances have introduced more specialized and dynamic classification methods. The Stakeholder-Specific Vulnerability Categorization (SSVC) system, developed by Carnegie Mellon University's SEI and CISA [33], offers a dynamic decision-making model with four primary categories (Track, Track*, Attend, and Act), based on five key values: exploitation status, technical impact, automation capability, mission prevalence, and public well-being impact. This work aligns with Seacord and

Householder's [132] vision of a structured approach that bridges theoretical taxonomy and practical application, while acknowledging that vulnerability classification represents just one component in the broader landscape of security research and practice.

### 2.2.3 Machine Learning Approaches in Vulnerability Classification

#### 2.2.3.1 Foundation and Early Developments

There has been a growing use of machine learning into vulnerability classification. Da Silva et al. [140] demonstrating how automated systems leveraging natural language processing and deep learning could address the limitations of traditional classification methods, particularly regarding manual classification subjectivity. Stehr and Kim [142] further advanced this field by introducing semantic vulnerability embeddings based on natural language processing techniques, establishing a foundation for enhanced risk assessment through clustering, classification, and visualization capabilities.

#### 2.2.3.2 Advanced Monitoring and Analysis

Recent applications of artificial intelligence have improved vulnerability classification approaches. Gajiwala [50] showed how AI enhances vulnerability assessment through continuous monitoring and dynamic analysis capabilities. Their research highlights AI's ability to process multiple data streams simultaneously, enabling a more comprehensive classification approach that considers network traffic, system logs, and threat intelligence feeds. This multi-dimensional analysis allows organizations to simulate various attack scenarios and prioritize vulnerabilities based on their potential impact. Ravikumar et al. [125] conducted an extensive evaluation of machine learning algorithms for vulnerability classification, comparing the effectiveness of various approaches including decision trees, support vector machines, and neural networks. Their research particularly focused on the role of feature selection techniques and model optimization strategies in improving classification accuracy. The combination of these studies illustrates how modern AI and machine learning approaches are enabling more sophisticated, accurate, and automated vulnerability classification systems that can adapt to evolving security threats while providing actionable insights for vulnerability management.

#### 2.2.3.3 Addressing Technical Challenges

A significant challenge in vulnerability classification stems from the inherent imbalance in security data, where certain types of vulnerabilities occur much more frequently than others. Ekle and Ulybyshev [40] addressed this critical issue through innovative data handling approaches. Their research revealed how imbalanced datasets can lead to biased classification models that excel at identifying common vulnerabilities but struggle to detect rare, potentially critical security threats. To overcome this limitation, they implemented the Synthetic Minority Over-sampling Technique (SMOTE), which generates synthetic examples of underrepresented vulnerability classes to create a more balanced training dataset. This approach proved highly effective, improving categorization model accuracy by 20% and demonstrating how sophisticated data handling can enhance the identification of less common but potentially devastating vulnerabilities. Their work underscores the importance of addressing data imbalance in machine learning-based vulnerability classification systems, as failing to identify rare vulnerabilities could leave systems exposed to sophisticated attacks that exploit these overlooked security weaknesses.

### 2.2.4 Integration of Classification Frameworks in CVE Analysis

Our study benefits from synthesizing multiple classification approaches to provide a comprehensive understanding of vulnerability relationships and lifecycles. The integration of established frameworks creates a multi-dimensional analytical approach that enhances our understanding of the interconnections between CVEs, CWEs, CPEs, exploits, and patches. Frei's temporal classification framework [46] provides the foundation for analyzing vulnerability lifecycles, particularly in understanding the progression from discovery to public disclosure. This temporal perspective is crucial when examining the relationships between CVEs and their associated exploits, as it helps identify patterns in vulnerability exploitation timelines. The CWE system's hierarchical structure [94] complements this by offering detailed categorization of weakness types, enabling us to analyze how different vulnerability classes correlate with exploitation patterns and patch development. Building upon NIST's technical vulnerability categories [99], we can better understand how different types of vulnerabilities (design, implementation, and operational) manifest across various platforms and systems. This classification is particularly valuable when analyzing the relationship between CVEs and CPEs, as it helps identify platform-specific vulnerability patterns. The recent machine learning approaches, such as those developed by Turtiainen and Costin [153], enhance our ability to automatically categorize and analyze these relationships at scale. For the Microsoft-specific patch analysis, we leverage the SSVC framework's [33] prioritization model, which considers factors like exploitation status and technical impact. This approach is particularly valuable when examining the relationship between CVE severity and patch release timing. By combining this with Da Silva et al.'s [140] catalog of security architecture weaknesses, we can better understand how different vulnerability types influence patching priorities and timelines. The synthesis of these classification frameworks enables our study to provide deeper insights into:

1. The correlation between vulnerability types (CWEs) and their exploitation patterns.
2. Platform-specific vulnerability trends through CPE analysis.
3. The relationship between vulnerability characteristics and patch development timelines in Microsoft's ecosystem.
4. Global trends in vulnerability discovery, disclosure, and exploitation.

This integrated approach aligns with Seacord and Householder's [132] vision of bridging theoretical taxonomy with practical application, while leveraging modern machine learning techniques like those demonstrated by Ekle and Ulybyshev [40] for enhanced analysis accuracy.

## 2.3 Vulnerability Lifecycle

The vulnerability lifecycle represents a sequence of events from initial discovery through remediation, encompassing multiple critical phases that impact overall security postures. This section examines the timeline and interconnected stages of vulnerability evolution, supported by empirical research and industry findings.

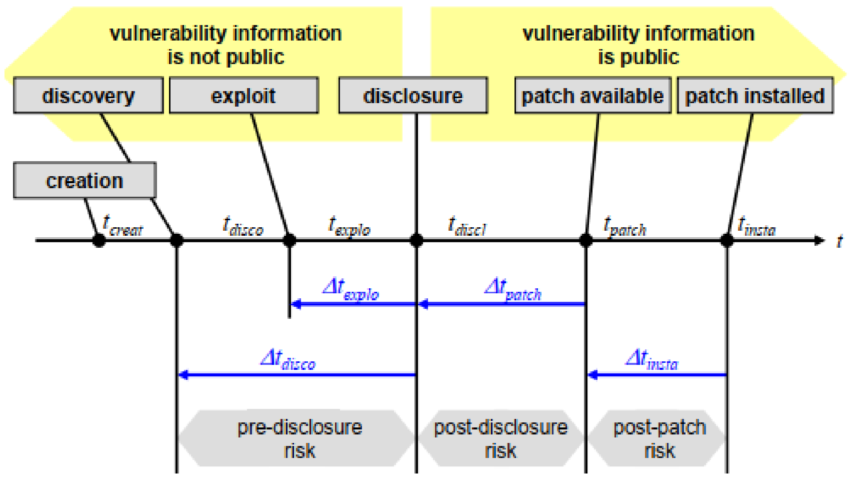


FIGURE 2.1: Vulnerability Lifecycle (adapted from Frei, 2009 [46])

Frei's seminal work established the foundational framework for understanding vulnerability lifecycles [46], identifying five distinct phases: discovery, disclosure, exploitation, patch availability, and patch deployment. The exact sequence of these events depends on the flow of the processes explained in the security ecosystem. This model has been useful in understanding the temporal relationships between these events, though recent research has revealed additional complexities in these relationships [130].

### 2.3.1 Creation, Discovery and Disclosure Phase

The discovery phase begins when a vulnerability is first identified, either by security researchers, malicious actors, or automated tools. Ruohonen's analysis demonstrates significant variations in the time between discovery and public disclosure, with critical vulnerabilities often facing longer assessment periods [130]. This creates a paradoxical situation where the most severe vulnerabilities may have extended exposure windows. Our preliminary analysis of 246,422 CVE records appears to support Ruohonen's findings, revealing a median disclosure time of 41 days and a substantial mean of 141.9 days, with a high standard deviation of 349.1 days indicating significant variability in disclosure times. Notably, our data shows that high-severity vulnerabilities indeed face longer assessment periods (median 54 days) compared to low-severity ones (median 13 days), aligning with Ruohonen's observations about extended exposure windows for critical vulnerabilities. An important temporal aspect often overlooked is the vulnerability creation time – the point at which the vulnerability is introduced into the software. Research by Ozment and Schechter 'Milk or Wine' [114] shows that many vulnerabilities are created during the initial development of software systems, with some remaining undiscovered for years after their introduction. This 'milk or Wine' hypothesis suggests that older code tends to have fewer undiscovered vulnerabilities, as the most serious flaws are typically found and fixed over time.

Research by MITRE's CVE team has shown that the discovery-to-disclosure timeline has been shortening in recent years, particularly for vulnerabilities in widely-deployed software [111]. However, this acceleration presents new challenges for vendors and security teams, who must rapidly assess and respond to newly disclosed vulnerabilities while maintaining accuracy in their evaluations.

The time between the discovery and public disclosure of software vulnerabilities is influenced by several factors, including the complexity of the vulnerability, the disclosure policies in place, and the efficiency of databases like the National Vulnerability Database (NVD). Complexity and severity, as measured by metrics such as the CVSS score, can affect how quickly a vulnerability is addressed and disclosed, as more complex vulnerabilities may require more time to understand and patch effectively [124]. Disclosure policies play a crucial role by setting a protected period for vendors to develop patches. These policies can either expedite or delay disclosure depending on how they are structured, with shorter protected periods generally pushing vendors to release patches more quickly [15]. The NVD, while a critical resource, has been shown to have delays in disclosure, with most vulnerabilities being delayed by 1-7 days, suggesting that reliance solely on the NVD may not always be the most efficient approach for timely disclosure [127]. Additionally, the time lag between discovery and exploitation, as discussed by Anand et al. [10], highlights the importance of timely disclosure to prevent exploitation, as any delay can increase the risk of vulnerabilities being exploited by malicious actors. Overall, these factors underscore the need for a balanced approach to vulnerability disclosure that considers both the urgency of addressing vulnerabilities and the practicalities of developing effective patches.

### 2.3.2 Exploitation Phase

The exploitation phase represents a critical period in the vulnerability lifecycle when discovered vulnerabilities become actively targeted by malicious actors. This phase typically occurs after vulnerability discovery and/or disclosure [46] but before patch deployment, characterized by a finite time lag during which attackers develop and refine their exploitation techniques [123]. Research by Allodi [8] has provided insights into exploitation patterns, revealing that vulnerability exploitation follows a heavy-tailed distribution, where a small subset of vulnerabilities accounts for the majority of observed attacks in the wild. This finding has significant implications for prioritizing vulnerability management efforts.

The evolution of exploitation dynamics has been dramatically influenced by the emergence of automated tools and exploit sharing platforms. Cyentia Institute's research demonstrates that vulnerabilities with published exploit code are seven times more likely to be exploited in the wild, with exploitation likelihood increasing from 3.7% to 37.1% when exploits are weaponized [66]. The timing of exploitation can vary significantly across different contexts, as demonstrated by Marconato et al. [92] in their comprehensive analysis of vulnerability life cycles. Their research shows that exploitation timing patterns differ substantially based on factors such as the target operating system, the complexity of the vulnerability, and the availability of exploit code. Our findings in Chapters 4 and 5 provide an opportunity to compare these claims against our dataset. In Chapter 4, our timeline analysis of exploited CVEs reveals a significant evolution in vulnerability disclosure and exploitation patterns. The data demonstrates a marked shift from predominantly pre-CVE exploits in early years (1999-2004, over 80%) to post-CVE exploits in recent years (2015-2024, over 80%). This temporal shift suggests improvements in coordinated vulnerability disclosure practices, though it doesn't directly address Cyentia's findings regarding exploit availability and exploitation likelihoodâĂŤan area we identify for future research with addition of GitHub Advisory data containing PoC information (Proof of Concept which represents the Exploitation code). Additionally, our analysis of Microsoft-specific vulnerabilities in exploitation patterns from 2016 to 2024 reveals that exploits are typically published approximately 168 days after CVE creation 5, which differs from the more immediate exploitation timeline suggested by [66] report. Furthermore, our analysis of high-severity vulnerabilities having longer exploitation timelines (median 141 days) than medium-severity ones (median 138 days) aligns with Marconato's observations about exploitation timing being influenced by vulnerability complexity, as severity often correlates with technical complexity. These comparisons highlight both consistencies and variations between our Microsoft-focused dataset and the broader vulnerability landscape, suggesting that vendor-specific factors may significantly influence exploitation patterns.

Key Insights from Microsoft's Digital Defense Report 2022 [103]

**Rapid Vulnerability Exploitation:**

- According to the **Microsoft Digital Defense Report 2022**, the timeframe between the announcement of a vulnerability and its exploitation has significantly decreased. On average, it takes only **14 days** for a vulnerability to be actively exploited in the wild after public disclosure.
- This accelerated exploitation timeline highlights the increasing efficiency of nation-state and criminal threat actors in weaponizing newly disclosed vulnerabilities.
- The report emphasizes the need for organizations to **prioritize immediate patching** of newly disclosed vulnerabilities and to adhere to coordinated vulnerability disclosure practices to mitigate risks effectively.

This finding, combined with the temporal modeling work of Rajasooriya et al. [123], emphasizes the critical importance of rapid patch deployment and the increasingly narrow window available for effective remediation. Understanding these temporal patterns and exploitation dynamics has become crucial for developing proactive security measures and maintaining effective vulnerability management strategies.

#### 2.3.2.1 Evolution of Exploit Information Sources

The landscape of exploit information has evolved significantly over time, shifting from traditional centralized databases to more dynamic and distributed platforms. This evolution has important implications for vulnerability research and management.

##### 2.3.2.1.1 Traditional Exploit Databases

Our primary data source, ExploitDB [112], represents one of several established platforms for exploit information. The ecosystem of traditional sources includes:

- **Exploit Database (ExploitDB)** [112]: Provides our core dataset of public exploits and vulnerable software, offering a structured and verified collection of exploit information.
- **Metasploit Framework** [101]: While primarily a penetration testing tool, Metasploit serves as a valuable repository of exploit modules, offering additional context through practical implementation examples.
- **National Vulnerability Database (NVD)** [111]: Supplements exploit information with comprehensive vulnerability data, providing crucial context for understanding exploit development and impact.

These traditional sources, while valuable, have limitations in terms of timeliness and completeness. This has led to the emergence of new platforms for sharing exploit information, particularly GitHub, which has become increasingly important in the security research community.

##### 2.3.2.1.2 GitHub's Emerging Role in Exploit Research

The shift toward GitHub as a significant source of exploit information represents a fundamental change in how vulnerability information is shared and accessed.

This evolution in exploit sharing platforms has important implications for vulnerability research and management. Recent work by Akhoundali et al. [4] provides further evidence of GitHub's crucial role, presenting findings on the platform's value for tracking and analyzing security fixes. While their work on CVE fix commits falls outside our current study's immediate scope, it highlights promising directions for future research in vulnerability lifecycle analysis.

The changing landscape of exploit information availability and the increasing importance of prediction has led to the development of more sophisticated approaches to vulnerability assessment and prioritization. One of the most significant developments in this area is the Exploit Prediction Scoring System (EPSS).

#### 2.3.2.2 The Exploit Prediction Scoring System (EPSS)

The growing complexity of vulnerability management and the need for empirically-based prioritization methods necessitated the development of improved risk assessment frameworks. Jacobs et al. [68] introduced the Exploit Prediction Scoring System (EPSS) to address limitations in existing vulnerability scoring methodologies.

##### 2.3.2.2.1 The development of EPSS

The EPSS framework, developed in 2019, represented a methodological shift from deterministic scoring systems toward probabilistic models that estimate exploitation likelihood within a 30-day timeframe post-disclosure. The researchers employed the following methodological approach:

1. **Data Collection**: The team aggregated vulnerability data from CVE and NVD, supplemented by exploit data from multiple sources including Proofpoint and Fortinet. This dataset covered vulnerabilities from June 2016 to June 2018.
2. **Feature Selection**: Recognizing that data quality significantly impacts prediction accuracy, the team implemented a three-stage filtering process:

  - Elimination of variables with complete separation (i.e., variables that perfectly predict exploitation or non-exploitation, causing statistical issues like infinite coefficient estimates and unreliable models).
  - Removal of variables observed in less than 1% of vulnerabilities
  - Removal of variables observed in less than 1% of vulnerabilities
  - Expert review to identify and remove potential bias sources

3. **Model Development**: The initial model employed logistic regression with Elastic net regularization (a statistical method that combines L1 and L2 regularization to improve model performance, enhance feature selection, and reduce multicollinearity issues). The model carefully selected 16 key variables including:

   - Vendor information
   - Exploit code availability
   - Vulnerability tags
   - Reference count

#### 2.3.2.2.2 EPSS Evolution

The system's evolution has been driven by real-world feedback and emerging challenges. Version 2, described in [67], introduced several significant improvements:

- Expansion to 1,164 features from the original 16
- Transition to XGBoost for improved modeling capabilities
- Reduction of the prediction window to 30 days to align with typical remediation cycles
- Implementation of a centralized architecture for better scalability

Version 3 further enhanced the system's capabilities, achieving an 82% improvement in classifier performance over version 2. These advancements demonstrate the rapid evolution of exploit prediction methodologies and their increasing importance in vulnerability management.

This progression in prediction capabilities has inspired other approaches to vulnerability assessment, including the development of Expected Exploitability (EE) metrics, which offer a complementary perspective on exploitation risk.

### 2.3.2.3 Expected Exploitability (EE)

Building on the foundations laid by EPSS, Suciu et al. [143] introduced the Expected Exploitability metric, representing another significant advance in vulnerability risk assessment. Unlike previous static approaches, EE introduced a dynamic perspective that evolves as new information becomes available.

#### 2.3.2.3.1 Key Innovations in EE

The development of EE addressed several critical challenges in vulnerability assessment:

- **Label Noise Management**: The researchers developed novel techniques to handle both class-dependent and feature-dependent noise in vulnerability data.
- **PoC Integration**: Unlike previous approaches that often dismissed Proof-of-Concept exploits, EE leveraged these as valuable indicators of exploitation potential.
- **Dynamic Updates**: The system continuously updates risk assessments as new information becomes available, providing more timely and accurate predictions.

With these improvements EE achieved 86% precision compared to 49% for previous approaches, which the authors attributed, at least partially, to incorporating dynamic data and sophisticated analysis techniques in vulnerability assessment.

#### 2.3.2.4 Metrics and Real-World Impact

The development of systems like EPSS and EE has led to a deeper understanding of critical vulnerability metrics that inform our research:

- **Time-to-Exploit (TTE)**: Measures the period between vulnerability disclosure and first exploitation, a critical metric in our analysis of Microsoft vulnerabilities.
- **Time-to-Patch (TTP)**: Tracks the duration from disclosure to patch availability, helping us understand vendor response effectiveness.
- **Time-to-Remediation (TTR)**: Encompasses the complete timeline from identification to full mitigation, providing context for overall vulnerability management effectiveness; however, this metric is challenging to measure accurately due to the lack of comprehensive data sources providing full remediation timelines.

These metrics serve as key indicators in our analysis of vulnerability lifecycles and inform our understanding of security response effectiveness. Their application to real-world scenarios has revealed important patterns in vulnerability management and exploitation, leading us to examine specific empirical studies that have shaped our understanding of these dynamics.

### 2.3.3 Patch Development and Deployment Phase

The patch development and deployment phase represents a critical juncture in the vulnerability lifecycle, characterized by complex technical and organizational challenges. Recent studies indicate that 71% of IT security professionals report patching processes as overly complex and time-consuming [65], leading to extended vulnerability windows even after patches become available.

A comprehensive analysis by Woo et al. [161] reveals significant challenges in patch effectiveness, demonstrating through large-scale empirical research that many security patches require subsequent modifications or improvements to achieve their intended protection goals. This finding aligns with the research of Xiong et al. [162], who established that the quality and timing of vulnerability information disclosure directly impacts vendors' patch development capabilities and outcomes.

The deployment phase presents several interconnected challenges that organizations must navigate:

- Organizational constraints and testing requirements, which must balance security needs with operational stability
- System dependencies and compatibility concerns, particularly in environments with shared code bases [70]
- Resource allocation and prioritization challenges, which Roytman and Jacobs [129] identify as frequently underestimated by organizations
- Operational impact considerations that affect deployment scheduling and implementation strategies

Cormack and Leverett [30] provide important insights into the regulatory landscape, demonstrating that while legal frameworks aim to encourage effective vulnerability response, their success has been limited. Their research suggests that organizations need to adopt more nuanced, risk-based approaches to vulnerability management that consider both compliance requirements and operational realities.

**Legal Frameworks in Vulnerability Management (Cormack and Leverett [30])**

- **Liability Laws**: The paper examines how liability laws aim to hold parties accountable for failing to manage software vulnerabilities effectively. However, these laws have shown limited success in promoting robust vulnerability management practices.
- **Product Quality Regulations**: These regulations are designed to ensure that software products adhere to established quality and security standards. The analysis indicates that, despite their intent, these regulations often fall short in encouraging effective patch management and vulnerability response.
- **Patching Mandates**: While patching mandates require organizations to apply updates within specified timeframes, the study reveals that such mandates are frequently ineffective in ensuring timely and comprehensive vulnerability remediation.
- **Privacy Laws**: Recent legal cases under privacy laws, as discussed in the paper, underscore the potential of risk-based patching. These cases highlight a framework where system managers, executives, and regulators could converge on more effective vulnerability response strategies.

Overall, Cormack and Leverett [30] argue that although these legal frameworks are intended to enhance software security, they often fail to achieve their desired outcomes. The authors propose that a risk-based approach to patching may offer a more effective means of aligning the interests of various stakeholders in the realm of vulnerability management.

This complex interplay of technical, organizational, and regulatory factors underscores the importance of developing comprehensive patch management strategies. Understanding these dynamics is crucial for improving the efficiency and effectiveness of vulnerability remediation efforts, particularly in environments where rapid response to security threats is essential.

### 2.3.4 Recent Lifecycle Analysis Methods

Contemporary approaches to vulnerability lifecycle analysis have evolved significantly with the integration of machine learning, advanced analytics, and statistical modeling. These methods are transforming how organizations understand and predict vulnerability behavior throughout different lifecycle phases.

Khazaei et al. [75] pioneered the application of natural language processing to vulnerability assessment, developing a system that analyzes vulnerability descriptions to predict CVSS scores. Their approach processes unstructured text from vulnerability reports, extracting key features that indicate severity and impact. By examining linguistic patterns and technical indicators in these descriptions, their model achieved 99% accuracy in predicting CVSS scores, demonstrating that automated systems can reliably assess vulnerability severity throughout their lifecycle phases without requiring manual expert analysis. Building upon this foundation, Costa et al. [31] expanded the scope of automated analysis by developing a machine learning framework that predicts individual CVSS metrics with 87% accuracy.

Rajasooriya et al. [123] introduced sophisticated statistical models based on Markov chain theory to estimate the probability of vulnerabilities transitioning between different lifecycle stages. Their work highlighted a significant challenge in vulnerability lifecycle analysis: the prevalence of censored data, where complete lifecycle information is unavailable for many vulnerabilities. This censoring occurs when vulnerabilities are discovered but their creation date remains unknown, or when exploitation events go undetected. Their research demonstrated that traditional statistical approaches often fail to account for this censored data, potentially leading to biased estimates of vulnerability exploitation risks and lifecycle durations. Despite these challenges, their mathematical framework provided valuable insights into vulnerability evolution patterns, enabling more proactive risk management strategies.

Franca et al.[45] further advanced this field by introducing a structured framework that not only identifies vulnerabilities but also provides classification and correction recommendations throughout the software development lifecycle. Their approach addresses effectiveness gaps in traditional vulnerability management methods by integrating automated analysis directly into development workflows.

The integration of real-time threat intelligence has further enhanced lifecycle analysis through the Exploit Prediction Scoring System (EPSS) [68]. This system provides dynamic risk assessment based on multiple factors, enabling more accurate prediction of exploitation likelihood throughout a vulnerability's lifecycle.

The evolution of these analysis methods reflects a broader shift toward more sophisticated, data-driven approaches to vulnerability management. By combining machine learning capabilities with real-time threat intelligence, organizations can now better understand not just the current state of a vulnerability, but also predict its likely progression through different lifecycle phases. This improved understanding enables more proactive security measures and more effective resource allocation in vulnerability management efforts.

### 2.3.5 Vendor-Specific Considerations

The vulnerability lifecycle patterns vary across software vendors, influenced by their development processes, patch release cycles, and product ecosystems. These variations reflect differences in how vendors integrate security into their development and maintenance processes.

#### 2.3.5.1 Enterprise Software Security Approaches

Enterprise software vendors employ different approaches to vulnerability management based on organizational structure and available resources. SAP has implemented security testing throughout the software development lifecycle, as documented by Bachmann and Brucker [19]. Their analysis demonstrates how enterprise vendors can incorporate security considerations into development processes, potentially enhancing their security outcomes.

#### 2.3.5.2 Open Source versus Proprietary Development Models

Open-source and proprietary vendors exhibit different vulnerability response patterns. Research by Shahzad et al. [134] indicates that closed-source vendors such as Microsoft and Apple generally achieve shorter patch response times, likely due to their centralized resources and structured development environments. In contrast, open-source vendors often show different response patterns. These findings align with Temizkan et al. [149], who documented how legislative pressures significantly influence patch release behaviors, particularly in how open-source and proprietary vendors respond differently to vulnerability disclosures.

#### 2.3.5.3 Browser Vendors and Economic Incentives

The browser market presents a unique case study in vulnerability management. Firefox and Chrome have developed distinctive approaches influenced by the financial motivations of vulnerability discoverers, the developers of Firefox and Chrome browsers, respectively, have established bug bounty programs. These programs allow security researchers to report vulnerabilities directly to the companies in exchange for monetary rewards. This approach helps in identifying and patching vulnerabilities before they can be exploited maliciously [7]. These financial incentives significantly impact patch release behaviors and demonstrate how market forces shape vendor responses to security challenges. The effectiveness of these programs varies across vendors, influenced by their specific implementation approaches and reward structures.

#### 2.3.5.4 Major Vendor Response Patterns

Leading software vendors such as Microsoft, Oracle, and Adobe demonstrate distinct patterns in handling vulnerabilities [134] [135]. These patterns are influenced by multiple factors, including the risk level of vulnerabilities and the existence of common vulnerabilities across multiple products. The following are examples of the patterns by vendor to handle vulnerabilities as per [134]:

**Microsoft:**

- Quick Patching in Windows: Microsoft is noted for being quicker in patching vulnerabilities in its Windows products compared to its other offerings. This indicates a prioritization of their flagship operating system, likely due to its widespread use and critical nature in enterprise environments.

- Zero-Day Patches: Over 85% of vulnerabilities in Windows are patched on or before their disclosure dates, demonstrating a proactive approach to security. This is crucial in minimizing the window of opportunity for potential exploits.

**Adobe:**

- Focus on High-Risk Vulnerabilities: Adobe tends to prioritize patching high-risk vulnerabilities, which is a common practice among vendors to mitigate the most severe threats first. This approach helps in reducing the potential impact of vulnerabilities on users and systems.

**General Observations:**

- Closed-Source vs. Open-Source: Closed-source vendors like Microsoft and Adobe generally have more resources to allocate towards security, resulting in quicker patching times compared to open-source vendors. This is attributed to their profit-driven models and the need to maintain customer trust.
- Vendor-Specific Trends: Each vendor has unique trends in how they handle vulnerabilities, influenced by their product portfolios, customer bases, and security policies. For instance, Microsoft's focus on Windows reflects its critical role in their ecosystem, while Adobe's emphasis on high-risk vulnerabilities aligns with its focus on widely-used applications like Acrobat and Flash.

The empirical analysis reveals that vendors exhibit consistent patterns across different time periods and product lines, with responses shaped by both internal processes and external pressures.

#### 2.3.5.5 Cross-Product Vulnerability Coordination

Major software vendors face the additional challenge of managing vulnerabilities across multiple product lines [135]. Their responses must consider the complexity of coordinating security measures across diverse product ecosystems while maintaining consistent security standards. The empirical analysis shows that vendors with multiple product lines must develop sophisticated frameworks to address both product-specific and cross-product security issues.

#### 2.3.5.6 Impact on Security Strategies

The diversity in vendor approaches necessitates tailored security strategies that account for each vendor's unique characteristics. Organizations managing multi-vendor environments must understand these variations to develop effective security policies. This understanding becomes particularly crucial when dealing with vendors like Microsoft, Oracle, and Adobe, which have distinct approaches to vulnerability management [134].

This examination of vendor-specific considerations reveals the complex interplay between organizational structure, development processes, and security management approaches. The variations in how different vendors handle vulnerabilities, from major corporations like Microsoft and Oracle to browser vendors like Firefox and Chrome, demonstrate the need for a nuanced understanding of vendor-specific patterns in developing effective security strategies. As demonstrated by

Tenable's research [150], vendor-specific vulnerability ratings often need to be supplemented with additional context and data sources to provide accurate exploitation risk assessments.

### 2.3.6 Impact of Zero-Day Vulnerabilities

Zero-day vulnerabilities, which are security flaws that are exploited by attackers before the vendor becomes aware of them or releases a patch, represent a critical challenge in vulnerability lifecycle analysis, as they fundamentally alter traditional discovery-to-exploitation timelines. These vulnerabilities pose unique risks because they exist in a pre-discovery state where defenders have no prior knowledge or prepared mitigations. Recent research by Yogi [164] demonstrates that zero-day vulnerabilities can remain undetected for months or even years, creating sustained exposure windows that traditional lifecycle models struggle to account for.

The impact of zero-day vulnerabilities extends across multiple dimensions of cybersecurity. P.[1] [115] has documented how these vulnerabilities disproportionately affect critical infrastructure and enterprise systems, particularly when threat actors specifically target high-value assets. Their research shows that organizations managing critical infrastructure face an average exposure window of 120 days before discovering zero-day vulnerabilities, significantly longer than the standard vulnerability lifecycle phases.

When these vulnerabilities are eventually discovered and disclosed, the response timeline becomes critically compressed. Research on Microsoft products reveals an average time-to-patch of 7 days once disclosed [1], though this varies significantly based on severity and exploitation status. Carleton [28] argues that this compressed timeline creates unique challenges for both vendors and defenders, requiring rapid response capabilities that often exceed traditional patch management processes.

The evolution of zero-day vulnerability management has been marked by significant technological advancements. Garg [53] explores how artificial intelligence and machine learning technologies are being deployed to detect potential zero-day vulnerabilities before they can be exploited, potentially shortening the pre-discovery phase of these vulnerabilities. This research is complemented by Krishnan's [78] analysis of 2022's zero-day vulnerabilities, which revealed that:

- 80% of successfully exploited zero-days targeted previously unknown vulnerability classes
- Organizations with advanced detection capabilities identified zero-days an average of 43 days faster than those without
- The financial impact of zero-day exploits increased by 65% compared to traditional vulnerability exploits

Understanding these unique lifecycle patterns is crucial for effective vulnerability management, particularly given the increasing sophistication of attack methods and the growing complexity of software systems. The interaction between different lifecycle phases, and their impact on overall security postures, continues to evolve with the emergence of new technologies and attack vectors.

[1]Full name: P., Sreeja - Title: Securing Cyberspace: Navigating Zero-Day Vulnerabilities through Discovery, Disclosure Strategies, and Defence Mechanisms

## 2.4 Vulnerability Management

Modern vulnerability management has evolved from a traditional patch-deployment process into a sophisticated, multi-dimensional discipline requiring advanced approaches for assessment, prioritization, and remediation of security vulnerabilities. Initially, vulnerability management was a broad responsibility of leaders across various domains, focusing on infrastructure and the anticipation of potential threats to maintain productivity and security [44]. Over the past two decades, the understanding of vulnerability has expanded to encompass multiple facets, including social, physical, spatial, systemic, and dynamic dimensions, as highlighted by European and international research [88]. In the realm of cybersecurity, vulnerability management has become increasingly critical due to the rising number of vulnerabilities, with a 55% increase noted over a five-year period, necessitating timely software updates and patches to protect systems from exploitation [49]. The conventional approach to vulnerability management, primarily centered on identification and patching, has proven inadequate for modern security challenges. Research by Garcia et al. demonstrates that effective vulnerability management requires dynamic adaptation capabilities and automated assessment mechanisms [52]. Their implementation of the Lazarus system showcases how machine learning approaches can be integrated into vulnerability management frameworks to achieve more precise risk assessment and prioritization algorithms. The development of comprehensive vulnerability management systems, which include modules for asset management, vulnerability scanning, and analysis, has improved the efficiency of managing vulnerabilities and assessing security conditions [58]. Furthermore, the use of ontologies in cybersecurity vulnerability management has facilitated the integration of intelligence from official sources and social media, enhancing the ability to issue alerts and manage vulnerabilities effectively [147]. The evolution of vulnerability management has also seen the establishment of systematic practices such as vulnerability scanning, patch management, and configuration management, which have become foundational to modern vulnerability management strategies [14]. As the field continues to advance, the focus on threat and vulnerability management emphasizes the importance of prioritizing risk assessments and filtering threat intelligence to manage operational risks effectively [160].

### 2.4.1 Vulnerability Assessment and Identification Frameworks

Recent developments in automated systems have reshaped how these frameworks are implemented. The HAL 9000 framework illustrates this progression, achieving noteworthy accuracy in CVSS score prediction through advanced natural language processing [47], effectively reducing the assessment delays associated with traditional manual processes. This improvement is particularly beneficial for organisations managing large volumes of vulnerabilities that require prompt assessment and prioritisation.

An important advancement in this area is the Skynet architecture, which introduces a cyber-aware, intrusion-tolerant overseer approach [48]. This architecture enhances vulnerability assessment by integrating multiple authoritative data sources within a unified analysis framework. The system processes comprehensive vulnerability information from the NVD database [111], correlates it with real-world exploitation data from ExploitDB [112], incorporates vendor-specific security advisories, and continuously monitors real-time threat intelligence feeds. This multi-source integration allows for more accurate threat assessment and automated response capabilities compared to traditional single-source approaches.

The integration of these frameworks enables organisations to develop comprehensive vulnerability mapping, linking potential attack patterns to specific weaknesses, thereby facilitating a more nuanced risk assessment and mitigation strategy [138]. However, effectively implementing these frameworks presents several challenges. The vast number of new vulnerabilities reported each year can overwhelm conventional manual analysis methods, resulting in delays and potential inaccuracies in vulnerability management [6].

To address these challenges, automated tools such as VulnBERTa have been designed to improve the accuracy and speed of linking CVEs to CWEs and CPEs [153]. Despite these technological advancements, the complexity of CVE texts and the frequent emergence of new technical terminology necessitate continuous updates and improvements in automated systems to maintain satisfactory accuracy levels [62]. Moreover, the absence of detailed attack semantic information in CVE descriptions can impede the precise characterisation of threats, making it challenging for organisations to prioritise vulnerabilities effectively [2].

### 2.4.2 Prioritization and Risk Assessment

Organizations employ sophisticated approaches to prioritize vulnerability remediation by integrating multiple factors, including CVSS scores, exploit availability, and business impact, while considering the temporal aspects of CVE lifecycles. The Common Vulnerability Scoring System (CVSS) serves as a fundamental metric for assessing vulnerability severity, though it often lacks the ability to predict exploitation likelihood, which is crucial for effective prioritization [20], [156]. To address this limitation, the security community has developed advanced prediction systems. The Exploit Prediction Scoring System (EPSS) [67] represents a significant advancement, predicting the probability of vulnerability exploitation within a 30-day timeframe. The EPSS, enhanced by community-driven insights and machine learning, significantly improves the prediction accuracy of exploit likelihood, enabling organizations to focus their resources on vulnerabilities most likely to be targeted [116]. Recent innovations have further refined vulnerability scoring mechanisms. The VIEWSS (Variable Impact-Exploitability Weightage Scoring System) provides additional refinement by adjusting the impact-exploitability score ratio, offering a more nuanced approach to scoring vulnerabilities based on their potential impact and exploitability [136]. Automated systems have become increasingly crucial in managing the complexity of vulnerability prioritization. The Automated Context-aware Vulnerability Risk Management (ACVRM) system [97] customizes patch prioritization based on an organization's specific context and risk appetite. This automation helps organizations optimize their vulnerability management processes by considering:

- Organization-specific risk profiles and tolerance levels
- Historical vulnerability exploitation patterns
- System criticality and exposure levels
- Resource availability for remediation efforts

These approaches collectively highlight the importance of integrating multiple factors in vulnerability prioritization. The combination of exploit prediction capabilities [67], business impact analysis [136], and temporal considerations [97] enables organizations to develop more effective vulnerability management strategies. This comprehensive approach ensures that organizations can allocate their resources efficiently while addressing the most critical security risks in a timely manner.

### 2.4.3 Patch Management Strategies

Practitioners and security teams employ diverse and sophisticated strategies for patch management, particularly for Microsoft systems, to effectively address the complex relationships between CVEs, exploits, and patch availability. A critical foundation of these strategies involves developing comprehensive patch management plans that encompass the complete lifecycle: from identification and prioritization through acquisition, installation, and verification of patches across the organization. This systematic approach has become essential due to the increasing reliance on technology and the accelerating pace of vulnerability exploitation [141]. The evolution of patch management has seen significant advancement through automation. Automated patch management systems have gained prominence as they substantially reduce the need for human intervention while increasing the success rate of patch deployments. These systems incorporate sophisticated review mechanisms that automatically adjust patch prioritization based on cumulative patches and their dependencies [98]. This automation has proven particularly valuable in large-scale environments where manual patch management would be impractical. The human element remains crucial, particularly in risk assessment processes. System administrators play a vital role by actively engaging in online communities to gather and synthesize information about patches, which informs their decision-making regarding patch application timing and prioritization [69]. This community-driven intelligence helps organizations anticipate potential deployment challenges and develop more effective patching strategies. Technical integrity in patch management has also seen significant improvements through the implementation of dual electronic signatures [76]. This security measure ensures patch authenticity during transmission, preventing forgery or tampering and guaranteeing that clients receive legitimate patches. Such verification mechanisms have become increasingly important as patch-related attacks have grown more sophisticated. Organizations are increasingly adopting hybrid patching policies that balance multiple considerations, including elapsed time since vulnerability disclosure and patch severity levels [36]. These policies help organizations optimize their approach by balancing the operational costs of patching against the security risks of delays. The integration of game theory with the Common Vulnerability Scoring System (CVSS) has further enhanced this decision-making process by modeling attacker-defender scenarios, leading to more accurate and effective defense strategies [90]. The overall effectiveness of these strategies depends heavily on their alignment with organizational contexts and risks. Patch management has evolved beyond simple technical implementation to become a crucial component of organizational risk management, requiring careful consideration of business continuity, resource allocation, and security priorities. This evolution reflects the growing understanding that effective patch management requires a balanced approach that considers both technical and organizational factors.

### 2.4.4 Temporal Aspects of Vulnerability Management

The temporal phases of vulnerabilities are managed through a sophisticated combination of automated systems, prioritization models, and coordinated disclosure practices, each playing a crucial role in the vulnerability lifecycle. The CAVP (Context-Aware Vulnerability Prioritization) model represents a significant advancement in this domain, automatically capturing the temporal characteristics of CVEs and prioritizing them based on context-aware scores that integrate seamlessly into organizational risk management workflows [71]. The automation of patch management has revolutionized how organizations handle temporal aspects of vulnerabilities. Modern automated patch management systems significantly reduce human intervention requirements while increasing the efficiency and success rate of patch deployments [98]. These systems are particularly effective at managing the critical time window between vulnerability discovery and patch deployment, helping organizations minimize their exposure to potential exploits. Machine learning technologies have transformed the temporal management of vulnerabilities by automating both identification and classification processes, as mentioned later in section 2.6.

The analysis of CVE lifecycle patterns has highlighted the critical importance of coordinated vulnerability disclosure (CVD) in effective vulnerability management. Research demonstrates that proper CVD practices are essential for timely patch deployment and reducing exploitation risks [118]. Organizations have begun prioritizing their patching efforts based on empirical data regarding exploit publication likelihood, aligning their response strategies with observed CVE lifecycle patterns [133]. To enhance the effectiveness of temporal management, organizations have implemented normalization frameworks for vulnerability severity scores across multiple databases [3]. This normalization ensures consistency in prioritization and enables more effective patch deployment strategies, particularly in cloud environments where rapid response is crucial. This comprehensive approach to temporal management helps organizations maintain an effective balance between security and operational stability.

### 2.4.5 Metrics and Measurement

Organizations employ a sophisticated array of metrics to measure the effectiveness of their vulnerability management programs, focusing on both quantitative and qualitative aspects of security posture. At the foundation of these measurements lies the Common Vulnerability Scoring System (CVSS), which provides a standardized base metric for assessing vulnerability severity based on exploitability and impact factors. However, organizations have recognized that CVSS alone is insufficient, leading to the development of more comprehensive measurement approaches that consider temporal and environmental characteristics to provide more accurate assessments within specific organizational contexts [74]. Risk-based metrics have emerged as a critical component of modern vulnerability measurement frameworks. These metrics incorporate the probability of compromise and evaluate the overall vulnerability of networks, with methodologies often refined through the Analytic Hierarchy Process (AHP) to reflect impacts on global infrastructure [158]. This evolution in measurement approaches acknowledges that vulnerability assessment must account for both technical severity and business context. Organizations have also implemented a three-tiered indicator system to comprehensively evaluate their security programs: Key Performance Indicators (KPIs), Key Risk Indicators (KRIs), and Key Goal Indicators (KGIs). This integrated approach helps organizations align their security posture with strategic objectives while maintaining operational effectiveness [106]. The framework enables organizations to:

- Track operational efficiency through KPIs
- Monitor emerging threats through KRIs
- Assess progress toward security objectives through KGIs

Statistical correlation measurements and Tier ratings derived from the NIST Cybersecurity Framework have become instrumental in assessing trends and evaluating the effectiveness of risk management processes [165]. These measurements provide organizations with quantifiable data to support decision-making and resource allocation in vulnerability management.

Performance measurement of vulnerability detection tools has evolved to include sophisticated benchmarking approaches using metrics such as precision and recall [12]. However, practitioners increasingly recognize that alternative metrics may be necessary depending on specific scenarios and organizational requirements. This adaptability in measurement approaches ensures that organizations can effectively evaluate their vulnerability management programs within their unique operational contexts. The timeliness of patch application has emerged as a critical metric, with organizations closely monitoring the time between vulnerability discovery and patch deployment [49]. This temporal measurement helps organizations identify bottlenecks in their patch management processes and assess their overall security responsiveness.

### 2.4.6 Challenges in Vulnerability Management

Organizations face increasingly complex challenges in vulnerability management, particularly due to the interconnected nature of modern security frameworks and tools. A fundamental challenge lies in the diversity and complexity of vulnerabilities themselves, which complicates both detection and classification processes. Traditional binary classification approaches have proven insufficient for capturing the nuanced characteristics of different CWE categories, leading to oversimplification and reduced effectiveness in vulnerability detection. Recent research by Atiiq et al. [16] demonstrates that utilizing CWE-specific classifiers can significantly improve detection accuracy by focusing on the unique characteristics of each vulnerability type. The quality and consistency of vulnerability reporting present another significant challenge across the global cybersecurity community. To address this, advanced machine learning models such as VulnBERTa have been developed to automate the assignment of CWE-related information to vulnerabilities [153] as discussed in 2.6. This automation helps standardize vulnerability data quality and improves the consistency of analysis across different organizations and security teams. Time management in patch deployment has become increasingly critical as the window between vulnerability disclosure and exploitation continues to shrink. Research indicates that many organizations struggle to recognize patching as a necessary business cost, resulting in inadequate patch management strategies [93]. This challenge is compounded by the growing sophistication of exploitation techniques and the increasing number of systems requiring protection. To address these complex challenges, the security community has developed several innovative solutions. Automated repair systems like CRepair [89] utilize advanced AI techniques to enhance vulnerability remediation, while context-aware models such as ACVRM and CAVP optimize the prioritization process [71], [97]. These automated solutions help organizations manage the overwhelming volume of vulnerabilities while maintaining accuracy in assessment and remediation. The scattered nature of vulnerability and exploit information across multiple repositories poses additional challenges for automation efforts. The AMADEUS-Exploit framework, developed by Varela-Vaca et al. [155], addresses this complexity by implementing feature models to systematically consolidate and organize vulnerability data from disparate sources. This framework enhances the vulnerability management process through formalized representation of system components and their security characteristics, enabling organizations to perform more precise impact assessments and prioritization. By creating structured relationships between system features and associated vulnerabilities, the framework facilitates more accurate classification and identification of potential security weaknesses while improving cross-team communication between security specialists, developers, and management. The model-based approach provides a standardized methodology for vulnerability data integration that accommodates the heterogeneous nature of security information, helping organizations establish a more comprehensive and consistent view of their security landscape across complex technological ecosystems.

Looking toward future solutions, the field is increasingly focused on integrating AI-driven techniques to bridge traditional static analysis with advanced detection methods [152]. These emerging approaches aim to:

- Optimize tools for diverse programming languages
- Incorporate predictive threat analysis
- Enhance software security in increasingly complex threat landscapes
- Improve automated response capabilities

The effective management of these challenges requires a multifaceted approach that combines improved data quality, automated tools, and strategic patch management to enhance the overall effectiveness of vulnerability management practices.

### 2.4.7 Commercial Solutions in Vulnerability Management

Vulnerability management in commercial solutions involves a multifaceted approach that integrates various tools and methodologies to identify, assess, and remediate vulnerabilities in software and network systems. Commercial solutions such as Nessus, F-Secure, and Qualys are widely used for vulnerability scanning and management, although they often lack transparency in their prioritization procedures, which can be a limitation for organizations seeking detailed insights into vulnerability criticality [157]. These tools typically automate the scanning process, generating vulnerability events and sorting them based on a grading criterion to facilitate efficient remediation [163]. This automation and standardization of vulnerability management processes have become increasingly crucial as organizations face growing numbers of security threats across increasingly complex IT environments.

#### 2.4.7.1 Analysis of Commercial Vulnerability Management Systems: The Tenable Case

Tenable's vulnerability management approach represents one example of commercial solutions in this domain, with its Nessus Professional product serving as its primary scanning technology. As documented by Tenable [150], the company has developed a methodology focused on continuous assessment rather than periodic scanning cycles. This approach reflects a broader industry shift toward more persistent monitoring techniques that aim to address the limitations of traditional vulnerability scanning intervals. At the core of Tenable's vulnerability management framework lies its proprietary Vulnerability Priority Rating (VPR) system, which addresses limitations in traditional CVSS-based assessments. Unlike standard scoring systems, VPR dynamically adjusts vulnerability priorities based on active exploitation patterns, threat intelligence, and asset criticality [166]. This approach enables organizations to focus their remediation efforts on vulnerabilities that pose the most immediate and significant risks. The platform's architecture facilitates comprehensive coverage across diverse IT environments. Its scanning capabilities encompass traditional infrastructure, cloud deployments, and containerized environments, providing unified visibility into an organization's security posture. This broad coverage is particularly crucial as organizations increasingly adopt hybrid infrastructure models [155]. Tenable's implementation of predictive prioritization represents a significant advancement in vulnerability management automation. The system employs machine learning algorithms to analyze multiple data points, including:

- Real-time threat intelligence
- Exploit availability
- Historical attack patterns
- Asset exposure levels
- Business context

This data-driven approach helps organizations reduce their mean time to remediation (MTTR) by automatically identifying high-risk vulnerabilities that require immediate attention [158]. The platform's integration capabilities with security orchestration and automated response (SOAR) systems further enhance its effectiveness in enterprise environments. Recent analysis indicates that organizations using Tenable's predictive prioritization capabilities experience a 97% reduction in the number of vulnerabilities requiring immediate attention, compared to traditional CVSS-based prioritization methods [74]. This efficiency gain allows security teams to focus their resources on the most critical issues while maintaining comprehensive vulnerability oversight. The solution's effectiveness in large-scale deployments is particularly noteworthy. In enterprise environments, Tenable's platform has demonstrated the ability to process and prioritize thousands of vulnerabilities daily, with automated workflows reducing manual assessment requirements by up to 85% [106]. This automation capability is crucial for organizations dealing with increasing vulnerability volumes and complexity. Through its integration of advanced analytics and machine learning, Tenable exemplifies the evolution of commercial vulnerability management solutions from simple scanning tools to comprehensive security platforms. Its approach to vulnerability prioritization and management offers valuable insights into how organizations can effectively balance comprehensive security coverage with practical resource constraints in modern IT environments.

## 2.5 Economics of Vulnerability Management and Lifecycle

The economics of vulnerability management represents a complex interplay between market forces, security considerations, and organizational decision-making. Anderson's seminal work on the economics of cybersecurity [11] established a fundamental framework for understanding how financial incentives drive vulnerability exploitation and management decisions. This economic perspective reveals how market dynamics influence both the development of exploits and the timing of patch releases, particularly for vulnerabilities that remain unpatched for extended periods.

The vulnerability lifecycle is significantly influenced by economic factors, with monetary incentives playing a crucial role in shaping both legitimate and underground markets. As Algarni [7] demonstrates, the increasing number of vulnerability discoverers motivated by financial rewards has created a dynamic marketplace that impacts both the software industry and broader society. This economic ecosystem has led to the emergence of complex market structures where exploit prices in underground markets often match or exceed rewards offered by legitimate bug-bounty programs [7].

The timing and effectiveness of vulnerability exploitation can be analyzed through statistical modeling approaches. Rajasooriya et al. [123] [122] have developed models that enable the estimation of risk levels and inform strategic patch deployment before vulnerabilities are exploited. These models are particularly valuable given the expanding nature of cybercrime markets and the growing number of market participants, which correlates with increased attack probabilities [9].

Marconato et al. [91] provide insights into how the vulnerability lifecycle, from initial discovery through to patching, can be modeled using probabilistic approaches to assess security risks and guide patching strategies. This understanding is crucial for organizations making economic decisions about resource allocation for vulnerability management. Kumar and Temizkan [81] further highlight how vendors prioritize vulnerability patching based on the potential impact on system confidentiality and integrity, demonstrating the economic considerations in patch development and deployment.

Recent statistical analysis by Brilhante et al. [27] has enhanced our understanding of the relationship between economic factors and vulnerability exploitation. Their work, combined with Anderson's economic framework [11], provides a comprehensive view of how market forces influence the vulnerability lifecycle. This integration of economic and statistical perspectives reveals the complex decision-making processes organizations face when managing vulnerabilities and deploying patches.

The economic landscape of vulnerability management continues to evolve, with the expansion of both legitimate and underground markets influencing exploitation patterns and patch development strategies. This economic reality underscores the importance of understanding market dynamics when developing effective vulnerability management strategies and highlights why certain vulnerabilities become more attractive targets for exploitation. The interplay between these economic factors and security considerations remains crucial for organizations seeking to optimize their vulnerability management resources and protect against potential exploits.

## 2.6 Predictive Modeling Advances

The evolution of predictive modeling in vulnerability lifecycle and management represents a significant advancement in cybersecurity research, marked by the integration of sophisticated machine learning approaches and comprehensive data analysis techniques. This section examines the progression from traditional vulnerability assessment methods to modern predictive frameworks, highlighting key developments and their implications for security practice.

### 2.6.1 Historical Development and Foundation

According to a comprehensive systematic literature review [21], the evolution of vulnerability prediction modeling can be traced through four distinct phases, each characterized by significant advancements in computational approaches. The initial exploration phase (2000-2010) marked the field's transition from traditional statistical methods to early machine learning applications. During this period, researchers like Neuhaus et al. [110] began exploring basic machine learning techniques for vulnerability prediction, establishing the groundwork for future advancements in automated vulnerability assessment. The expansion phase (2010-2015) witnessed a significant broadening in the application of machine learning techniques. This period was characterized by extensive experimentation with various algorithms, including decision trees, support vector machines, and random forests. Notable work by Zimmermann et al. [170] and Shin and Williams [139] focused on addressing fundamental challenges in vulnerability prediction, particularly the persistent issue of imbalanced datasets, while also introducing more sophisticated feature selection methods and early ensemble learning approaches. A transformative shift occurred during the deep learning emergence phase (2015-2020), marked by the introduction and rapid adoption of neural network architectures for vulnerability prediction. Studies by Perl et al. [119] and Hovsepyan et al. [60] exemplified this evolution by advancing model capabilities, particularly in processing large-scale vulnerability datasets and identifying complex patterns. Sabottke et al. [131] demonstrated further progress by successfully integrating diverse data sources, including social media data, in their exploit prediction model. The consolidation phase (2020 onwards) has focused on refining and optimizing both machine learning and deep learning approaches. This period has been characterized by the maturation of prediction techniques, with researchers like Dam et al. [35] developing more sophisticated models capable of predicting both vulnerability probability and severity. The introduction of just-in-time (JIT) vulnerability prediction [5] further revolutionized the field by enabling real-time vulnerability detection during code development. Notable advancements include Brilhante et al.'s [27] implementation of heavy-tailed distributions for modeling vulnerability timelines, and the development of transfer learning techniques for cross-language vulnerability prediction. A significant milestone emerged in 2024 with Turtiainen and Costin's [153] introduction of VulnBERTa, a cybersecurity-focused model based on the RoBERTa transformer architecture. This innovation automated the assignment of Common Weakness Enumerations (CWEs) to vulnerability descriptions, substantially improving open cybersecurity data quality and achieving higher classification granularity. Concurrent research by Elbes et al. [41] explored AI and ML's comprehensive potential in vulnerability management, demonstrating enhanced threat detection accuracy and response strategies. Kumari and Sharma [82] developed early fuzzer attack detection systems using logistic regression, Gaussian naive Bayes, and LSTM networks. This period has also seen significant improvements in model evaluation practices, with researchers employing more comprehensive performance measures and standardized evaluation methodologies.

### 2.6.2 Diverse Machine Learning Applications

The field has witnessed multiple successful applications of various ML techniques. The evolution of vulnerability classification has benefited from automated analysis approaches, enabling more efficient identification and categorization of security weaknesses. Tanga et al. [148] made significant advances in vulnerability classification by implementing a comprehensive framework that combines multiple ML approaches. Their work integrated Bayesian Networks for probabilistic modeling of security characteristics, ensemble techniques like XGBoost for handling heterogeneous data, and Variational Autoencoders (VAEs) for zero-day vulnerability detection. They further enhanced the framework with reinforcement learning for dynamic threat identification and mathematical techniques such as Markov Chains and Lasso Regression to improve model interpretability. Their unsupervised learning approach, utilizing Isolation Forests and Gaussian Mixture Models, proved particularly effective for anomaly detection in network packets.

### 2.6.3 Advanced Computational Models and Machine Learning Integration

Recent years have seen substantial advances in computational approaches to vulnerability prediction. Freitas et al. [47] demonstrated that Random Forest algorithms can achieve 99% accuracy in predicting CVSS scores using natural language processing on vulnerability descriptions. Their HAL 9000 framework [48] represents a significant advancement by incorporating real-time threat intelligence and historical exploitation patterns, enabling more accurate predictions of both vulnerability severity and exploitation likelihood.

The integration of deep learning approaches has particularly transformed the field. Liu et al. [89] developed the VulPCL framework, utilizing BLSTM and CodeBERT for vulnerability prediction, categorization, and localization, achieving significant improvements in accuracy across multiple datasets. The adoption of transfer learning techniques has also addressed the challenge of limited code samples across different programming languages [57], expanding the applicability of these models across diverse development environments.

### 2.6.4 Multi-technique Approaches and Dataset Considerations

The effectiveness of vulnerability prediction models has been significantly enhanced through the integration of multiple techniques. Kande [73] demonstrates that combining software metrics with text mining and automated static analysis improves the precision, recall, and F-measure of vulnerability prediction models. This multi-technique approach has been particularly successful when applied to large-scale infrastructures and complex environments, as shown by Avadanei et al. [17].

Dataset quality and comprehensiveness have emerged as crucial factors in model development. The VALIDATE repository [42] addresses the challenge of dataset availability and reliability, providing a centralized resource for researchers. Le et al. [85] have shown that including latent vulnerabilities, often overlooked in traditional datasets, can significantly enhance prediction models by increasing the number of detectable vulnerabilities and improving overall accuracy.

### 2.6.5 Current State and Emerging Technologies

The current landscape of vulnerability prediction is characterised by sophisticated approaches that combine multiple analytical methods with varying prediction targets and data sources. Neural networks, particularly convolutional and recurrent types, have demonstrated effectiveness in processing structured and sequential data such as network traffic and logs, enabling rapid anomaly detection and response to threats rather than explicitly predicting vulnerability timelines [154]. These approaches focus on real-time detection of vulnerabilities as they manifest, utilising adaptive learning and reinforcement learning for dynamic threat mitigation.

SecScore enhances the traditional CVSS by incorporating empirical evidence of exploit code development, allowing for a more dynamic and timely assessment of vulnerabilities [72]. The MTLPT framework exemplifies the use of multi-task learning models to predict and identify potential vulnerabilities in software, specifically focusing on minority categories containing critical vulnerabilities [63]. This framework processes real-world vulnerability data, which is highly imbalanced, utilising custom lightweight Transformer blocks and position encoding layers to capture long-range dependencies and complex contextual information from source code. Rather than predicting time-to-vulnerability, MTLPT emphasises improving sensitivity and accuracy in identifying rare but severe vulnerability types through learning latent relationships between different vulnerability categories.

Looking forward, the integration of emerging technologies presents both opportunities and challenges for vulnerability prediction. Issues of model robustness and interpretability necessitate the development of adversarial training and explainable AI approaches to improve the transparency and reliability of AI-driven cyber defence systems [154]. The rise of AI-powered cyber threats, alongside vulnerabilities in IoT ecosystems and cloud computing, underscores the need for multifaceted mitigation strategies beyond simple prediction models [126].

The adoption of Zero Trust Architecture for access control verification, blockchain for data integrity, and behavioural analytics for monitoring user patterns and identifying deviations represents the next frontier in vulnerability prediction and management [77]. Behavioural analytics, in particular, shows promise for predicting potential vulnerabilities by analysing user behaviour patterns in system access logs and other operational data. These advanced approaches, whilst not directly predicting time-to-next-vulnerability, offer complementary strategies that address different aspects of vulnerability management, from prevention to detection and mitigation. Nevertheless, these advancements require ongoing innovation and adaptation to address the complexities of modern cybersecurity challenges, particularly as the underlying data used for modelling evolves in quantity, quality, and dimensionality.

## 2.7 Gaps in Current Research

The comprehensive review of vulnerability lifecycle and management literature reveals several significant areas requiring further investigation. These gaps represent critical opportunities for advancing our understanding of vulnerability management and improving security practices.

### 2.7.1 Temporal Risk Assessment

Current approaches to vulnerability management demonstrate significant limitations in addressing temporal aspects of risk assessment. While Frei's seminal work

[46] established the foundational framework for understanding vulnerability lifecycles, subsequent research has struggled to fully capture the dynamic nature of vulnerability risks over time. As discussed earlier in 2.6. The statistical modeling approaches developed by Rajasooriya et al. [123], while groundbreaking in their application of Markov chain theory, continue to face significant challenges with incomplete vulnerability data, particularly in cases where complete lifecycle information is unavailable. These censoring issues, combined with limited incorporation of zero-day exploit dynamics, create substantial uncertainty in temporal risk assessments and hinder the development of more accurate predictive models.

The integration of temporal factors in risk assessment becomes particularly crucial when examining the relationship between patch availability and actual deployment. Research by Dissanayake [37] highlights a critical gap in understanding how risk levels evolve during this transition period, particularly in enterprise environments where patch deployment often faces significant delays. This gap becomes especially pronounced when considering the findings of Akhoundali et al. [4], which demonstrate the complex relationship between vulnerability disclosure and patch deployment timelines.

Contemporary research efforts, while advancing our understanding of vulnerability lifecycles, have not fully addressed the challenge of quantifying risk evolution over time. The work of Brilhante et al. [27] underscores this limitation, particularly in contexts where multiple patches may be required to fully remediate a vulnerability. This gap in temporal risk assessment becomes especially critical when considering the findings of Turtiainen and Costin [153], which highlight the need for more sophisticated approaches to vulnerability classification and risk assessment over time.

### 2.7.2 Dynamic Severity Scoring

The limitations of current severity scoring mechanisms represent a significant gap in vulnerability management research. Traditional approaches to vulnerability scoring, particularly the Common Vulnerability Scoring System (CVSS), remain predominantly static and fail to capture the evolving nature of vulnerability risks. While EPSS [68] has made substantial progress in predicting exploitation likelihood, research by Xiong et al. [162] reveals significant limitations in current scoring systems' ability to reflect real-world exploitation patterns.

The disconnect between initial severity scores and actual exploitation patterns becomes particularly evident when examining vendor-specific vulnerabilities. Research by Shahzad et al. [134] demonstrates how current scoring systems struggle to account for vendor-specific characteristics and exploitation patterns. This limitation becomes more pronounced when considering the findings of Gandhi et al. [51], which highlight the challenges in maintaining accurate severity assessments throughout a vulnerability's lifecycle.

Recent research efforts, particularly those focused on Microsoft vulnerabilities, underscore the need for more dynamic scoring approaches. The work of Dissanayake [37] reveals how current scoring systems fail to capture the temporal evolution of vulnerability severity, especially in cases where initial assessments may not accurately reflect long-term exploitation risks.

This gap becomes particularly significant when considering empirical evidence from our dataset, which demonstrates the limitations of current scoring systems in predicting actual exploitation patterns. Our analysis of CVE lifecycle events by severity levels in chapter 5 reveals counterintuitive findings that challenge conventional vulnerability prioritisation approaches. Specifically, Critical vulnerabilities exhibited a median time of 192 days between CVE reservation and exploit availability, substantially longer than High (141 days) and Medium severity vulnerabilities (138 days). This contradicts the assumption that higher-severity vulnerabilities are necessarily exploited more rapidly.

Further, our examination of 415 Microsoft-related CVEs that were both exploited and patched showed that whilst the exploit-to-patch relationship varies by severity, higher CVSS scores did not consistently correlate with faster exploitation timeframes. For instance, Critical vulnerabilities had a mean time difference of 227.68 days between CVE creation and exploit availability, compared to 147.75 days for Low severity vulnerabilities-a pattern that challenges risk assessment frameworks that prioritise solely based on CVSS scores.

The severity distribution analysis of exploited vulnerabilities revealed additional complexity. Of the top 10 most common CWEs, SQL Injection (CWE-89) had the highest proportion of verified exploited CVEs (2,839 out of 11,356), surpassing more numerous vulnerability types like Cross-site Scripting (CWE-79), which had 1,896 exploited instances despite being the most prevalent weakness overall (26,728 CVEs). This suggests that exploitation likelihood depends more on the specific weakness type than on raw severity scores alone.

Our data also revealed that the relationship between patching and severity follows expected patterns only for non-exploited vulnerabilities, where Critical vulnerabilities had a median patching time of 54 days compared to 106 days for Low severity issues. However, for exploited vulnerabilities, this relationship inverts-with Critical exploited vulnerabilities showing the longest median time to patch at 186 days, a 71% increase compared to the overall patching timeline for Critical vulnerabilities. This substantial difference supports Costa et al.'s [31] assertion that conventional severity scoring systems have significant limitations in predicting real-world exploitation and remediation patterns.

### 2.7.3 Vulnerability Data Quality

The challenges in vulnerability data quality and consistency have been a persistent concern in cybersecurity research. While earlier work by Liao et al. [87] established foundational frameworks for vulnerability data collection and analysis, the evolving nature of security threats has introduced new complexities in data management and curation.

Research by Ruohonen [130] further exposed the temporal inconsistencies in vulnerability reporting. Their analysis revealed that while the median delay between CVE publication and CVSS scoring was minimal (1 day), the substantial standard deviation of 410.9 days indicated significant variations in reporting timelines. Building on this, Anwar et al. [13] conducted a comprehensive quality assessment of the National Vulnerability Database (NVD), uncovering systematic issues including erroneous publication dates, incorrect affected product names, and inconsistent severity scores.

The scope of data quality issues became more apparent through Guo et al.'s [56] detailed analysis, which revealed that approximately 56% of CVEs lack vulnerability type information, while 85% miss critical root cause details. This incompleteness in vulnerability documentation has significant implications for security assessment and mitigation strategies. Recent work by Sun et al. [145] further identified discrepancies across four large-scale, actively maintained vulnerability databases (NVD, IBM X-Force, ExploitDB, and Openwall) in key aspects such as severity scores, affected versions, and vulnerability types, highlighting the persistent challenges in maintaining consistent vulnerability reporting.

The integration of other sources as a vulnerability intelligence source highlighted early data quality challenges. Mittal et al. [109] demonstrated this through their analysis of 143,701 cybersecurity-related tweets, where only 7% contained sufficient technical information for vulnerability analysis. Their evaluation revealed that just 57.2% of tweets were correctly tagged by their concept extraction system, with 33.2% being completely misclassified. These findings emphasized the challenges of extracting structured vulnerability information from unstructured social data.

While advances in automated classification systems like VulnBERTa [153] have shown promise, fundamental data quality issues persist. Croft et al. [32] recently conducted a comprehensive analysis of four state-of-the-art software vulnerability datasets (Big-Vul, Devign, D2A, and Juliet), identifying significant data quality deficiencies. They found that 20-71% of vulnerability labels in real-world datasets are inaccurate, with D2A showing the highest inaccuracy at 71.4% and Devign the lowest at 20%. Additionally, they identified severe duplication problems, with 17-99% of data points being duplicated across datasets, particularly in D2A (97.9%) and Juliet (83.7%). These findings have significant implications for the reliability of vulnerability assessment models and tools, with evaluation performance decreasing by up to 82% after removing duplicates. Fortunately, our preliminary analysis does not employ any of these datasets, avoiding these potential data quality pitfalls.

**The vulnerability datasets examined in Croft et al. [32]**

The vulnerability datasets examined in Croft et al. [32] serve crucial purposes in the cybersecurity research community, particularly for developing and evaluating Software Vulnerability Prediction (SVP) models. Each dataset employs different methodologies for labelling vulnerable code:

- **Big-Vul:** Created by Fan et al. [43], this dataset contains 188,636 C/C++ functions (5.78% vulnerable) derived from security vendor-provided labels. Researchers use it to develop machine learning models that can identify vulnerabilities based on patterns observed in historically vulnerable code. It is primarily used by academic researchers and tool developers working on static analysis techniques.
- **Devign:** Developed by Zhou et al. [169], this dataset comprises 27,318 functions (45.61% vulnerable) with developer-provided labels extracted directly from GitHub commits. Security researchers and tool developers leverage this dataset to build models that can learn from actual developer-identified vulnerabilities, particularly for vulnerability identification using graph neural networks.
- **D2A:** Created by Zheng et al. [167], D2A contains 1,295,623 functions (1.44% vulnerable) with tool-created labels generated through differential analysis. This dataset is primarily used by researchers developing automated vulnerability detection methods that combine static analysis with machine learning approaches.
- **Juliet:** This synthetically created dataset by NIST [111] contains 253,002 functions (36.77% vulnerable) that demonstrate known vulnerable code patterns. Security tool vendors, academic researchers, and educators use Juliet as a benchmark to evaluate the effectiveness of static analysis tools and to teach secure coding practices.

These datasets are predominantly utilized by:

- **Academic researchers** developing new vulnerability detection algorithms and conducting empirical studies on vulnerability patterns
- **Security tool vendors** benchmarking and improving commercial and open-source static analysis tools
- **ML/AI researchers** exploring applications of machine learning to cybersecurity challenges
- **Software development organizations** enhancing their secure development lifecycle processes

The significant data quality issues identified by Croft et al. raise important concerns about the reliability of models trained on these datasets, particularly since many published research papers have reported performance metrics without addressing these underlying data quality problems.

These findings have significant implications for the reliability of vulnerability assessment models and tools. The work of Okutan et al. [113] has demonstrated some progress in addressing these challenges through automated curation approaches, achieving impressive F-measure values over 0.9 for vulnerability characterization and reducing CVE characterization time by up to 47

The integration of exploit information presents additional complexity in vulnerability data management. While ExploitDB offers valuable exploit availability data, research by Sabottke et al. [131] and Pastor et al. [117] highlights the ongoing challenges in establishing reliable connections between exploit information and vulnerability data. Recent work by Sun et al. [146] has proposed promising solutions through aspect-level matching techniques for recovering traceability across heterogeneous vulnerability databases, though challenges remain in standardizing reporting practices and ensuring consistent metadata across sources.

Kuehn et al. [80] have recently proposed the OVANA, or Overt Vulnerability source ANAlysis, which is a system that uses machine learning and natural language processing to analyze and enhance the information quality of vulnerability databases by improving the accuracy, completeness, and uniqueness of the data, resulting in a 51.23% improvement in these quality indicators on the National Vulnerability Database, despite the improvement mentioned, a recent study Sun et al. [144] have highlighted ongoing issues with inconsistent measurement and incorrect detection of software names in security vulnerability reports. These recent developments underscore both the progress made and the persistent challenges in maintaining high-quality vulnerability data across diverse sources and platforms.

### 2.7.4 Integration of Historical Patterns

Building upon the data quality challenges discussed in the previous section 2.7.3, the integration of historical vulnerability patterns presents additional complexities in understanding and predicting software security risks. The inconsistencies and incompleteness in vulnerability databases directly impact our ability to leverage historical data effectively for pattern analysis and prediction. This section examines these interconnected challenges and their implications for vulnerability research and practice.

The narrow focus of Automated Vulnerability Detection (AVD) research represents a fundamental challenge in utilizing historical patterns. Shereen et al. [137] identified that current approaches often overlook crucial areas for effective real-world vulnerability detection, including diversified problem formulations and broader language support. This limitation is further emphasized by Dzielski and Devine [39], who note that despite advancements in AI and vulnerability databases, there remains a critical gap in forecasting vulnerability occurrences, which is essential for preventing future security failures.

Empirical evidence highlights the importance of historical pattern analysis while revealing ongoing challenges in leveraging this data effectively. Maza and Sultana [96] demonstrated through their study of Apache Tomcat that specific software metrics significantly change over the vulnerability lifecycle, yet these insights are not fully integrated into current prediction models. Brilhante et al. [27] further illuminate these challenges, particularly in pattern recognition and historical data integration, while Garcia et al. [52] emphasize the difficulties in applying historical patterns to emerging vulnerability types.

The temporal aspects of vulnerability management add another layer of complexity. Iannone et al. [64] found that vulnerabilities often remain unfixed for extended periods, with developers typically introducing them during high workload phases. This finding aligns with Johnson et al. [70]'s research on how limited understanding of historical patterns impacts patch development and deployment strategies. Roytman and Jacobs [129] further elaborate on these challenges, particularly in applying historical insights to current vulnerability management practices.

The economic dimension of vulnerability lifecycles provides additional context. Algarni [7] investigated the relationship between vulnerability lifecycles and markets, revealing how financial incentives drive both the discovery and exploitation of vulnerabilities. Their research suggests that understanding and controlling these market dynamics could help mitigate lifecycle-related risks. This economic perspective is complemented by Gueye and Mell's [55] analysis of Common Vulnerability Scoring System (CVSS) metrics, which shows that while the vulnerability landscape has remained relatively stable over time, this historical consistency is underutilized in developing proactive security measures.

The Vulnerability Landscape Insights from Gueye and Mell's [55]

Gueye and Mell's [55] analysis revealed a remarkably stable vulnerability landscape over the 15-year period from 2005 to 2019. Despite substantial security efforts during this time, they found that the same fundamental characteristics consistently dominate:

- **Network exploitation:** *"The overwhelming majority of software vulnerabilities are exploitable over the network (i.e., remotely)."*
- **Low complexity:** *"The complexity to successfully exploit these vulnerabilities is dominantly low."*
- **Minimal authentication:** *"Very little authentication to the target victim is necessary for a successful attack."*
- **Limited user interaction:** *"Most of the flaws require very limited interaction with users."*

This historical consistency represents a significant **underutilized** opportunity in vulnerability management. As the authors note:

> *"Our findings are consistent to previous studies... This indicates that the same vulnerabilities are still being found in our software, suggesting that the community has not been doing a great job correcting the most common vulnerabilities."*

They even express concern about future progress, stating:

> *"We don't want to write this same paper 15 years from now showing that, once again, nothing has changed."*

While the security community has extensive data showing which vulnerability characteristics persistently dominate, this knowledge has not been effectively leveraged to prioritize remediation efforts or develop targeted defensive strategies. The authors suggest the security community needs to *"stop and think"* about software development approaches and vulnerability identification methods. This gap offers promising research directions for utilizing historical vulnerability patterns to develop more focused secure coding practices, create context-aware prediction models, and design frameworks specifically addressing these persistent vulnerability characteristics.

These gaps in historical pattern integration, compounded by the data quality issues discussed earlier, point to a pressing need for more comprehensive approaches that can effectively leverage historical data for vulnerability prediction and management. Future research should focus on developing frameworks that can better incorporate historical patterns while accounting for the evolving nature of software vulnerabilities, their economic implications, and the challenges in patch development and deployment.

## 2.8 Summary and Conclusions

This comprehensive review has examined the complex landscape of vulnerability lifecycle and management, revealing both significant advances in our understanding and critical areas requiring further investigation. The evolution of vulnerability research has been marked by increasingly sophisticated approaches to classification, analysis, and management, driven by the growing complexity of modern software systems and the escalating sophistication of cyber threats.

The foundational work of Frei [46] established crucial frameworks for understanding vulnerability lifecycles, while subsequent research by Rajasooriya et al. [123] and Brilhante et al. [27] has enhanced our understanding of temporal patterns in vulnerability evolution. The integration of machine learning approaches, particularly exemplified by recent work from Turtiainen and Costin [153], has transformed how we analyze and classify vulnerabilities, enabling more accurate and automated assessment processes.

The development of standardized frameworks, including CVE, CWE, and CPE systems, has provided essential structure for vulnerability management, though research by Martin et al. [94] and subsequent studies highlight ongoing challenges in maintaining consistency across these frameworks. The emergence of sophisticated prediction systems, such as EPSS [68], represents a significant advancement in our ability to anticipate and prioritize vulnerability risks, while work by Dissanayake [37] demonstrates the complex relationships between vulnerability disclosure and exploitation patterns.

Vulnerability management practices have evolved considerably, with research by Gandhi et al. [51] revealing the importance of integrated approaches that combine technical assessment with organizational context. The work of Woo et al. [161] has highlighted critical considerations in patch management, while studies by Xiong et al. [162] demonstrate the complex interplay between vulnerability disclosure and patch development processes.

Recent advances in automated vulnerability assessment, particularly through machine learning applications, have transformed traditional approaches to security management. The introduction of systems like VulnBERTa [153] has improved our ability to automatically classify and assess vulnerabilities, though significant challenges remain in ensuring consistent and accurate vulnerability data across different platforms and sources.

The economic dimensions of vulnerability management, as explored through the work of Shahzad et al. [134], reveal complex incentive structures that influence both vulnerability disclosure and patch development. These economic considerations become particularly significant when examining vendor-specific approaches to vulnerability management, as demonstrated by research focusing on Microsoft's security practices [37].

Despite these advances, significant gaps remain in our understanding of vulnerability lifecycles and management practices. Current approaches struggle to fully capture the temporal evolution of vulnerability risks, particularly in the critical period between patch availability and deployment. The limitations of static severity scoring systems, as highlighted by recent research, suggest the need for more dynamic approaches to risk assessment. Furthermore, challenges in data quality and consistency, along with limited integration of historical patterns, continue to impact our ability to effectively predict and manage vulnerability risks.

The emergence of zero-day vulnerabilities presents particular challenges, as demonstrated by the research of Yogi [164] and Adam [1], highlighting the need for more sophisticated approaches to vulnerability detection and response. The work of Akhoundali et al. [4] further emphasizes the importance of understanding patch effectiveness and the potential need for multiple fixes to address complex vulnerabilities fully.

This review demonstrates that while significant progress has been made in understanding and managing vulnerabilities, considerable work remains in developing more comprehensive and dynamic approaches to vulnerability assessment and management. The integration of machine learning techniques, combined with improved understanding of vulnerability lifecycles and exploitation patterns, offers promising directions for addressing current limitations in vulnerability management practices.

Through careful analysis of the literature, it becomes clear that effective vulnerability management requires a nuanced understanding of both technical and organizational factors. The continued evolution of threat landscapes and software complexity demands increasingly sophisticated approaches to vulnerability assessment and management, while challenges in data quality and consistency remind us of the fundamental importance of robust and standardized information sharing practices in the security community.

# CHAPTER 3

# METHODOLOGY

This chapter presents the methodological framework employed to investigate the Common Vulnerabilities and Exposures (CVEs) and its lifecycle phases, with particular focus on the patching phase for vulnerabilities affecting Microsoft products. The research adopts a systematic approach to data collection, processing, integration, and analysis to provide comprehensive insights into vulnerability characteristics, discovery patterns, exploitation timelines, and remediation processes. The methodology addresses several research questions regarding the temporal dynamics between vulnerability discovery, exploitation, and patching, while identifying key factors that influence these processes.

## 3.1 Research Design and Theoretical Framework

### 3.1.1 Research Approach

This study employs a mixed-methods research design by combining quantitative analysis of large-scale vulnerability datasets with qualitative interpretation of emerging patterns and trends. This methodology acknowledges that while empirical data provides valuable insights into vulnerability lifecycles, the complex socio-technical context in which vulnerabilities exist requires interpretive analysis to fully understand the implications. The research design follows a sequential process that begins with comprehensive data collection from multiple authoritative sources, followed by systematic data processing and integration. This foundation supports descriptive and exploratory analysis of vulnerability characteristics, temporal analysis of vulnerability lifecycles, comparative analysis of exploitation and patching patterns, and interpretation of findings within the broader cybersecurity context. This approach aligns with Mittal et al.'s [109] framework for transforming publicly available information into actionable intelligence through systematic collection, processing, and analysis. As described by Liao et al. [87], vulnerability intelligence represents one of the most structured and standardized domains within cybersecurity research. This standardization is reflected in our methodological framework, which establishes clear relationships between CVEs, their associated exploits, patches, and classifications through Common Weakness Enumeration (CWE) and Common Platform Enumeration (CPE).

### 3.1.2 Theoretical Underpinnings

The theoretical foundation of this research incorporates elements from several frameworks: Vulnerability Lifecycle Theory, as established by Frei [46], provides a fundamental framework for understanding the temporal dynamics of vulnerabilities from discovery to patching. Frei's model identifies five distinct phases: creation, discovery, disclosure, exploitation, and patch availability. Our research extends this model by incorporating more recent advances in vulnerability intelligence gathering and analysis techniques, particularly for Microsoft-related vulnerabilities. Security Economics concepts view security as an economic problem involving trade-offs between resource allocation and risk mitigation. As Anderson [11] demonstrated, financial incentives drive both vulnerability exploitation and management decisions, particularly for vulnerabilities that remain unpatched for extended periods. This economic perspective is essential for interpreting the patterns observed in exploitation timelines and patch development processes. Socio-Technical Systems Theory recognizes that vulnerability management involves complex interactions between technical systems, organizational processes, and human behaviors. This theoretical lens supports our mixed-methods approach, acknowledging that effective vulnerability management requires consideration of both technical vulnerabilities and the socio-organizational context in which they are addressed. As highlighted by Franca et al. [45], traditional vulnerability management approaches often contain fundamental effectiveness gaps when they fail to consider the socio-technical dimensions of security. These theoretical frameworks inform both the data collection strategy and the interpretation of findings, enabling a comprehensive understanding of the vulnerability lifecycle beyond purely technical considerations.

## 3.2 Data Sources and Collection Strategy

### 3.2.1 Primary Data Sources

The research leverages multiple authoritative data sources to ensure comprehensive coverage and enable cross-validation of information. Each source provides unique perspectives on different aspects of the vulnerability lifecycle.

#### 3.2.1.1 CVE V5 Database

The Common Vulnerabilities and Exposures (CVE) Version 5 database serves as the primary source of vulnerability information. It provides standardized identifiers and descriptions for publicly known cybersecurity vulnerabilities. Key attributes extracted from this source include:

- CVE identifiers and descriptions
- Publication, reservation, and update dates
- Severity ratings and CVSS scores
- Affected products and vendors
- Problem types and references

Data was collected from the official CVE GitHub repository [108], which contains the most up-to-date vulnerability information in JSON format. This approach aligns with Pastor et al.'s [117] findings on the importance of leveraging standardized, machine-readable vulnerability data for comprehensive security analysis. The CVE dataset provides a foundation for understanding the broader vulnerability landscape, with specific focus on the subset of vulnerabilities affecting Microsoft products. This dataset contains 246,422 unique CVE records, offering a comprehensive view of the vulnerability ecosystem across multiple vendors and products. Of these, 162,095 CVEs (65.8%) have at least one associated CWE, while the remainder lack specific weakness classification or have generic classifications such as 'NVD-CWE-noinfo' or 'NVD-CWE-Other'.

#### 3.2.1.2 ExploitDB Dataset

The Exploit Database (ExploitDB) [112] provides a comprehensive archive of public exploits and corresponding vulnerable software. This resource is maintained by Offensive Security and serves as a valuable source for understanding exploit availability and characteristics. Key attributes extracted include:

- Exploit identifiers and descriptions
- Publication and update dates
- Author information
- Target platforms and systems
- Verification status
- Associated CVE identifiers

Data was collected from the official ExploitDB GitLab repository, specifically focusing on the files_exploits.csv file, which contains comprehensive metadata for all publicly available exploits. This dataset encompasses 46,494 exploit records, providing insights into real-world exploitation patterns and timelines. These exploit records correspond to 22,130 unique CVEs, representing approximately 9% of all vulnerabilities in the CVE database. Among these, 19,392 CVEs (87.63%) have a single associated exploit, while 2,738 CVEs (12.37%) have multiple exploits, with some having as many as 38 exploits for a single vulnerability. Web application vulnerabilities constitute the most prevalent exploit type, with 9,937 verified and 2,583 unverified exploits in this category. The integration of exploit data supports the research by Sabottke et al. [131], who demonstrated that publicly available exploit information can be effectively leveraged to understand vulnerability exploitation risks and patterns. Additionally, as highlighted by Allodi [8], exploit datasets offer crucial insights into the heavy-tailed distribution of vulnerability exploitation, where a relatively small subset of vulnerabilities accounts for the majority of observed attacks.

#### 3.2.1.3 Microsoft Security Response Center (MSRC) API

The Microsoft Security Response Center (MSRC) API [104] provides detailed information about vulnerabilities affecting Microsoft products and the corresponding security updates. This source was particularly valuable for understanding the patching timeline for Microsoft-related vulnerabilities. Key attributes extracted include:

- Security update identifiers and release dates
- Affected products and versions
- CVE identifiers addressed by each update
- Threat and severity information

- Exploitation status and potential
- Remediation details

Data was collected through programmatic API calls to the MSRC service, retrieving Common Vulnerability Reporting Framework (CVRF) documents for each monthly security update since 2016. This approach aligns with Dissanayake's [37] research on the significance of vendor-specific vulnerability management practices in understanding security response effectiveness. The MSRC dataset provides information on 9,991 CVEs patched by Microsoft since 2016 (9,981 out of 9,991 are unique CVEs), enabling detailed analysis of Microsoft's patching practices and the effectiveness of their security response processes. Of the total Microsoft-related vulnerabilities in CVE-V5 DB (19,620 CVEs), 13,285 (68%) emerged after 2016, while 6,335 (32%) were identified before 2016. Notably, among the MSRC-documented patches, 124 CVEs (1.24%) were reported as actively exploited according to Microsoft's own tracking, while Exploit DB records show exploitation for 468 Microsoft-related CVEs, highlighting discrepancies in exploitation reporting.

#### 3.2.1.4 Common Weakness Enumeration (CWE) Database

The Common Weakness Enumeration (CWE) database provides a standardized taxonomy of software and hardware weaknesses. This resource helps categorize vulnerabilities based on their underlying weaknesses, enhancing the understanding of vulnerability patterns. Key attributes extracted include:

- CWE identifiers and names
- Weakness abstractions and descriptions
- Status information
- Related weaknesses

Data was collected from the official CWE List [107], incorporating NVD vulnerability database [111] mapping of CVEs to their corresponding CWEs to enable weakness-based analysis. The CWE dataset provides comprehensive classification for 593 unique weakness types, supporting the analysis of 175,250 CVE-CWE associations across 162,095 unique CVEs. This indicates that some CVEs are associated with multiple weaknesses, with up to 7 CWEs mapped to a single vulnerability in extreme cases. Analysis revealed that approximately 93% of vulnerabilities with CWE data have only one associated weakness type, while 6% have two associated CWEs.

This approach aligns with the work of Martin et al. [94], who established the importance of standardized weakness taxonomies for effective vulnerability management. Additionally, as noted by Meunier [102], the multi-dimensional nature of vulnerabilities requires careful consideration of their classification at different abstraction levels.

#### 3.2.1.5 Common Platform Enumeration (CPE) Database

The Common Platform Enumeration (CPE) database provides a standardized method for identifying applications, operating systems, and hardware devices. This resource helps in precisely identifying affected systems and components. Key attributes extracted include:

- CPE identifiers
- Vendor information
- Product details
- Version specifications

CPE data was primarily extracted from CVE records in the NVD database [111], which include CPE strings and configuration for affected products and systems. The CPE dataset encompasses information on approximately 2.5 million affected product instances (CPEs) across 32,572 distinct vendors and 146,783 unique products. Analysis of the CPE data revealed that among vulnerable systems, applications (designated with CPE part 'a') constitute the largest category with over 1,110,415 vulnerable instances, followed by operating systems (part 'o') with 899,967 instances, and hardware (part 'h') with 461,952 instances. Microsoft-related products appear prominently in the vulnerability landscape, with Microsoft ranking among the vendors with the highest exploitation rates (9.13%, representing 1,747 exploited CVEs (As per Exploit-DB)out of 19,139 total Microsoft CVEs (as per CVE-V5)). This comprehensive platform enumeration supports detailed analysis of vulnerability distribution across different systems and vendors, aligning with the work of Cheikes et al. [29] on standardized platform identification for security analysis.

### 3.2.2 Data Collection Process

#### 3.2.2.1 API Integration and Data Retrieval

Custom Python scripts were developed to interact with various APIs and repositories to retrieve the required data. The process involved: MSRC API Integration implemented authentication and pagination handling to retrieve CVRF documents for each month between 2016 and 2024. This approach focused on structured retrieval of Microsoft's security update information, ensuring comprehensive coverage of their patching activities. CVE Repository Cloning involved cloning the official CVE V5 GitHub repository to access JSON files containing vulnerability information. This method ensured access to the most current and comprehensive vulnerability data. ExploitDB Data Retrieval fetched the exploit metadata CSV file from the ExploitDB repository. This provided standardized access to public exploit information, supporting analysis of exploitation patterns and timelines. Data Validation implemented checks to ensure the completeness and integrity of retrieved data. This step was crucial for maintaining data quality and reliability throughout the analysis process. The API integration process was designed to be resilient against network failures and API rate limits, incorporating retry mechanisms and incremental data retrieval to ensure comprehensive data collection. This approach addresses challenges identified by Ruohonen [130] regarding data availability and consistency in vulnerability databases.

#### 3.2.2.2 Automated Collection Framework

To maintain data currency and enable reproducible research, an automated data collection framework was implemented. The framework consists of:

- Scheduled Data Collection using cron jobs configured to retrieve updated vulnerability information on a daily basis. This automation ensures that the dataset remains current throughout the research period.

- Delta Processing through scripts designed to process only new or updated records from each source, minimizing redundant processing. This approach optimizes resource utilization while maintaining data currency.
- Logging and Monitoring with comprehensive logging of all data collection activities and automated alerting for collection failures. This ensures the reliability and continuity of the data collection process.
- The data collection process is illustrated in Figure 3.1.

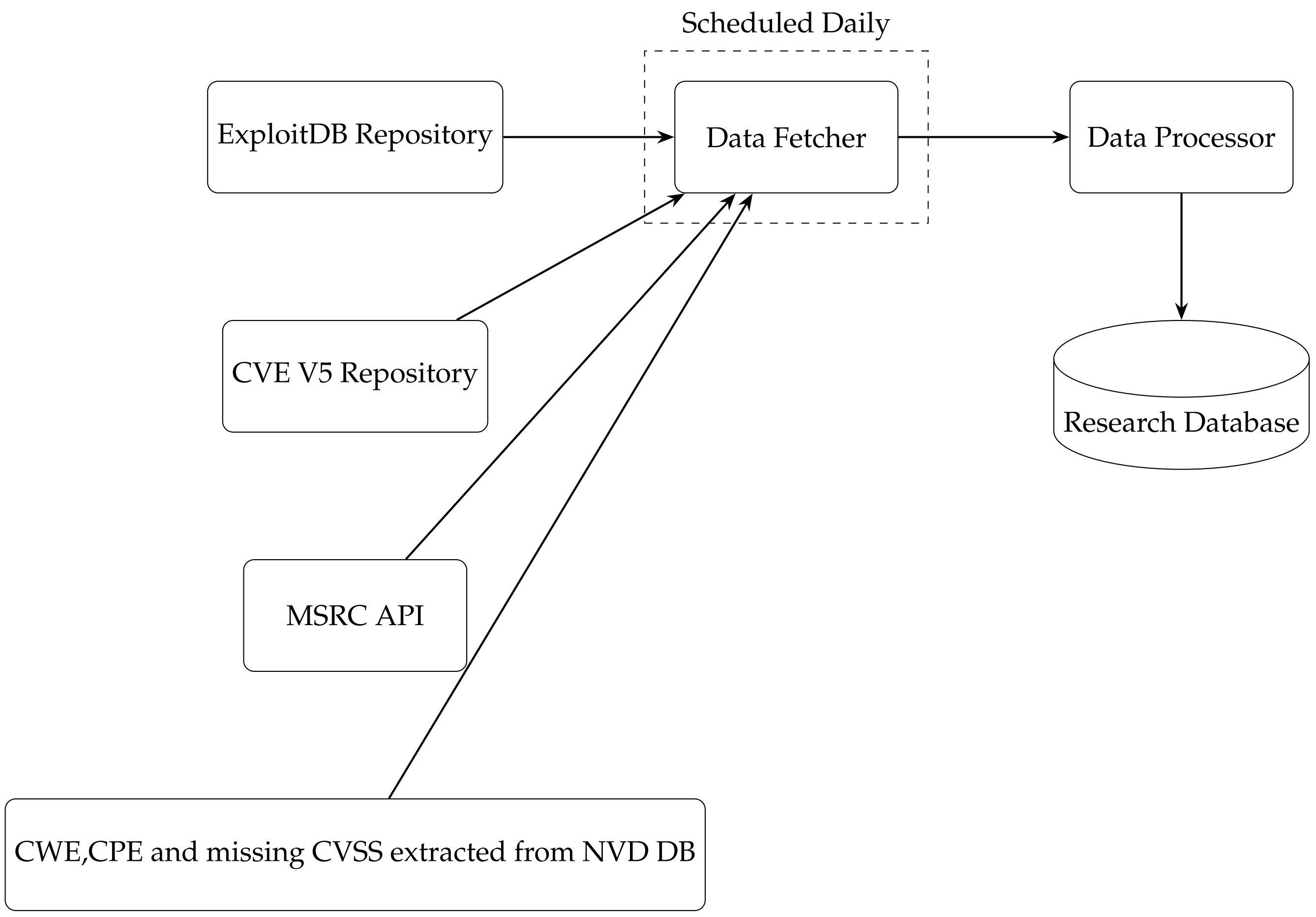


FIGURE 3.1: Automated Data Collection Framework

This automated framework ensures that the research dataset remains current throughout the duration of the study, capturing new vulnerabilities, exploits, and patches as they emerge. The approach aligns with recommendations by Gandhi et al. [51] regarding the importance of continuous data collection and processing in vulnerability management research.

## 3.3 Data Processing and Preparation

### 3.3.1 Data Cleaning and Preprocessing

The raw data collected from various sources required extensive cleaning and preprocessing to ensure consistency, accuracy, and analytical utility. The preprocessing pipeline involved several key stages:

- **Handling Missing Values** implemented strategies for addressing missing data, including null value imputation for CVSS scores and logical defaults for missing dates. This approach was particularly important for the CVE dataset, where approximately 79% of entries initially lacked CVSS scores. To address this, we leveraged additional data from the NVD database to supplement missing severity information, achieving CVSS coverage for 93% of vulnerabilities.
- **Date Standardization** converted all date fields to a consistent format (ISO 8601) and ensured proper timezone handling. This standardization was crucial for temporal analysis, enabling accurate calculation of time differences between key lifecycle events.
- **Text Normalization** standardized text fields such as descriptions, vendor names, and product identifiers to enable effective matching and analysis. This process improved the accuracy of entity recognition and relationship identification across different datasets.
- **Duplicate Removal** identified and resolved duplicate entries, particularly for CVEs referenced across multiple sources. This step ensured that each vulnerability was represented only once in the analytical dataset, preventing potential biases in frequency-based analyses.
- **Data Type Conversion** ensured appropriate data types for all fields, such as converting string-based scores to numeric values. This conversion supported mathematical operations and statistical analyses on numerical data.

Particular attention was paid to the CVE V5 data, which required complex JSON parsing and field extraction to transform the nested structure into a tabular format suitable for analysis. This processing challenge aligns with observations by Sun et al. [145] regarding the complexity of vulnerability data structures and the need for sophisticated processing techniques.

### 3.3.2 Feature Engineering

To enable comprehensive analysis of vulnerability lifecycles, several derived features were created:

- **Temporal Features** calculated time deltas between key events using earliest dates (e.g., time from CVE reservation to exploit publication, time from exploit to patch). These features enabled detailed analysis of the temporal relationships between different lifecycle phases, supporting insights into the efficiency of vulnerability management processes. As demonstrated in Chapters 4 and 5, these temporal features were crucial for understanding the evolution of exploitation patterns over time and the relationship between vulnerability disclosure, exploitation, and patching.
- **Severity Categorization** mapped raw CVSS scores to categorical severity levels (Low, Medium, High, Critical) based on standard thresholds. This categorization facilitated analysis of vulnerability distribution by severity and supported investigation of how severity influences different aspects of the vulnerability lifecycle. As shown in Chapter 5, these severity categories were essential for analyzing how different vulnerability severities affect exploitation and patching timelines.

- **Exploitation Status** created binary and categorical features indicating whether a vulnerability has been exploited and its verification status. These features were crucial for analyzing the relationship between vulnerability characteristics and exploitation likelihood, enabling the detailed exploitation pattern analysis presented in Chapter 4.
- **Microsoft Association** developed logic to identify vulnerabilities directly related to Microsoft products versus third-party software running on Microsoft platforms. As detailed in Chapter 5, this distinction enabled focused analysis of Microsoft's security practices while acknowledging the broader ecosystem in which their products operate, revealing significant differences in lifecycle patterns between Microsoft products and third-party components.
- **CPE Parsing** extracted vendor, product, and version information from complex CPE strings to enable more granular analysis. This parsing supported detailed investigation of vulnerability distribution across different platforms and systems, as presented in Chapter 4's analysis of affected platforms.
- **Product Family Grouping** implemented hierarchical categorization of Microsoft products to enable analysis at the product family level. As shown in the product family vulnerability analysis in Chapter 4, this grouping facilitated understanding of how vulnerabilities are distributed across Microsoft's product ecosystem, revealing concentration patterns in specific product families like Microsoft Office, Windows, and .NET.

These derived features enhanced the analytical capabilities of the dataset, enabling more nuanced exploration of vulnerability patterns and lifecycles. The approach aligns with recommendations by Xiong et al. [162] regarding the importance of contextual feature engineering in vulnerability analysis.

### 3.3.3 Dataset Integration

A critical aspect of the methodology involved integrating data from multiple sources into a cohesive analytical dataset. The integration process followed these steps:

- **Schema Alignment** mapped fields across datasets to establish common attributes (e.g., CVE identifiers, dates, severity measures). This alignment created a unified data structure that preserved the unique characteristics of each source while enabling cross-source analysis.
- **Hierarchical Merging** implemented a hierarchical merging strategy, starting with core CVE data and progressively enriching it with exploit information, patch details, and CWE classifications. This approach ensured that the integration process preserved the integrity of the original data while creating valuable new relationships. As demonstrated in Chapter 5, this merging strategy enabled complex three-point lifecycle analysis incorporating CVE reservation, exploit publication, and patch release data.
- **Conflict Resolution** developed rules for resolving conflicting information across sources, generally prioritizing more specific or recent data. This step addressed challenges identified by Anwar et al. [13] regarding inconsistencies in vulnerability datasets. As seen in Chapter 4's analysis of exploit verification status and Chapter 5's examination of patching information, this approach enabled consistent analysis despite initial data discrepancies.

- **Relationship Modeling** established relationships between entities (e.g., CVEs, exploits, patches, CWEs, CPEs) to enable relational queries and analyses. These relationships formed the foundation for investigating the interconnections between different aspects of the vulnerability lifecycle, particularly evident in Chapter 4's analysis of CWE pairs and triplets and Chapter 5's integrated CVE-Exploit-Patch lifecycle investigation.

- **Many-to-Many Handling** implemented specific processing workflows to handle many-to-many relationships between entities. As shown in Chapter 4's product family vulnerability analysis (Figure 4.24), this approach properly accounted for instances where a single CVE affects multiple products or product families, or where multiple CWEs are associated with a single vulnerability.

The integration process resulted in a comprehensive dataset containing 246,422 CVE records, enriched with exploit information (46,494 records) and Microsoft patch details (9,991 CVEs (9,981 Unique)). This integrated dataset enables analysis of the complete vulnerability lifecycle, from discovery through exploitation to patching. The approach aligns with recommendations by Pastor et al. [117] regarding the integration of multiple data sources for comprehensive vulnerability intelligence.

### 3.3.4 Creation of Specialized Datasets

#### 3.3.4.1 Microsoft CVE Subset

To support the research focus on Microsoft-related vulnerabilities, a specialized dataset was created by:

- Filtering CVEs based on vendor field in CPE identifier containing "Microsoft" or "microsoft".

- Augmenting with text-based matching of Microsoft products in the description field.

- Cross-referencing with the MSRC dataset to include all CVEs addressed in Microsoft security updates.

This process identified 19,620 Microsoft-related CVEs, with 6,335 (32%) occurring before 2016 and 13,285 (68%) emerging after 2016. This distribution reflects Microsoft's significant presence in the software ecosystem and highlights the importance of analyzing their security practices within the broader vulnerability landscape.

#### 3.3.4.2 Microsoft Product Family Dataset

For the analysis of vulnerability distribution across Microsoft product families, presented in Chapter 4, a specialized dataset was constructed through a systematic process:

- Primary data was sourced exclusively from the MSRC patched CVEs dataset (9,981 CVEs from 2016 to April 2024).

- This methodological choice was made after discovering that 3,054 of these patched CVEs (approximately 30.6%) had no associated CPE information in the CVE-V5 database.

- A many-to-many relationship model was implemented to account for vulnerabilities affecting multiple products or product families simultaneously.
- This approach resulted in 47,211 product-vulnerability pairs across 9,981 unique CVEs, enabling the comprehensive product family distribution analysis presented in Figure 4.24.

#### 3.3.4.3 CWE Distribution Dataset

For the analysis of Common Weakness Enumeration (CWE) distribution in Microsoft patches, presented in Chapter 4, a specialized dataset was created by:

- Merging the MSRC dataset with the comprehensive CVE-V5 database to obtain CWE information for patched vulnerabilities.
- Identifying that 4,607 CVEs (approximately 46.2%) from the MSRC patched dataset had no associated CWE information.
- Focusing analysis on the 5,374 CVEs with identifiable CWE classifications.
- Accounting for many-to-one relationships between CWEs and CVEs, as some CVEs are associated with multiple CWEs.

This approach enabled the CWE distribution analysis presented in Figure 4.26, highlighting the predominance of memory safety issues in Microsoft's vulnerability landscape.

#### 3.3.4.4 Exploited and Patched CVE Dataset

A key dataset for analyzing vulnerability lifecycles was created by identifying CVEs that had both associated exploits and patches. This dataset was constructed by:

- Merging CVE data with exploit information based on CVE identifiers.
- Further merging with MSRC patch data to identify patched vulnerabilities.
- Filtering to include only those CVEs with both exploitation and patch information.
- Categorizing CVEs as directly related to Microsoft products (391 CVEs, 94%) or related to third-party software operating on Microsoft platforms (24 CVEs, 6%).

This process resulted in a dataset of 415 CVEs that had been both exploited and patched, forming the basis for the detailed lifecycle analysis presented in Chapter 5. The differentiation between Microsoft-specific products and third-party components enabled the identification of significant differences in lifecycle patterns between these categories, particularly in the outlier analysis presented in section 5.4.1.

The data preparation and integration process for Exploited and Patched CVE dataset is illustrated in Figure 3.2. The CWE dataset creation mentioned in Chapter 4 Figure 4.26. The Microsoft product family dataset mentioned in Chapter 4 Figure 4.24.

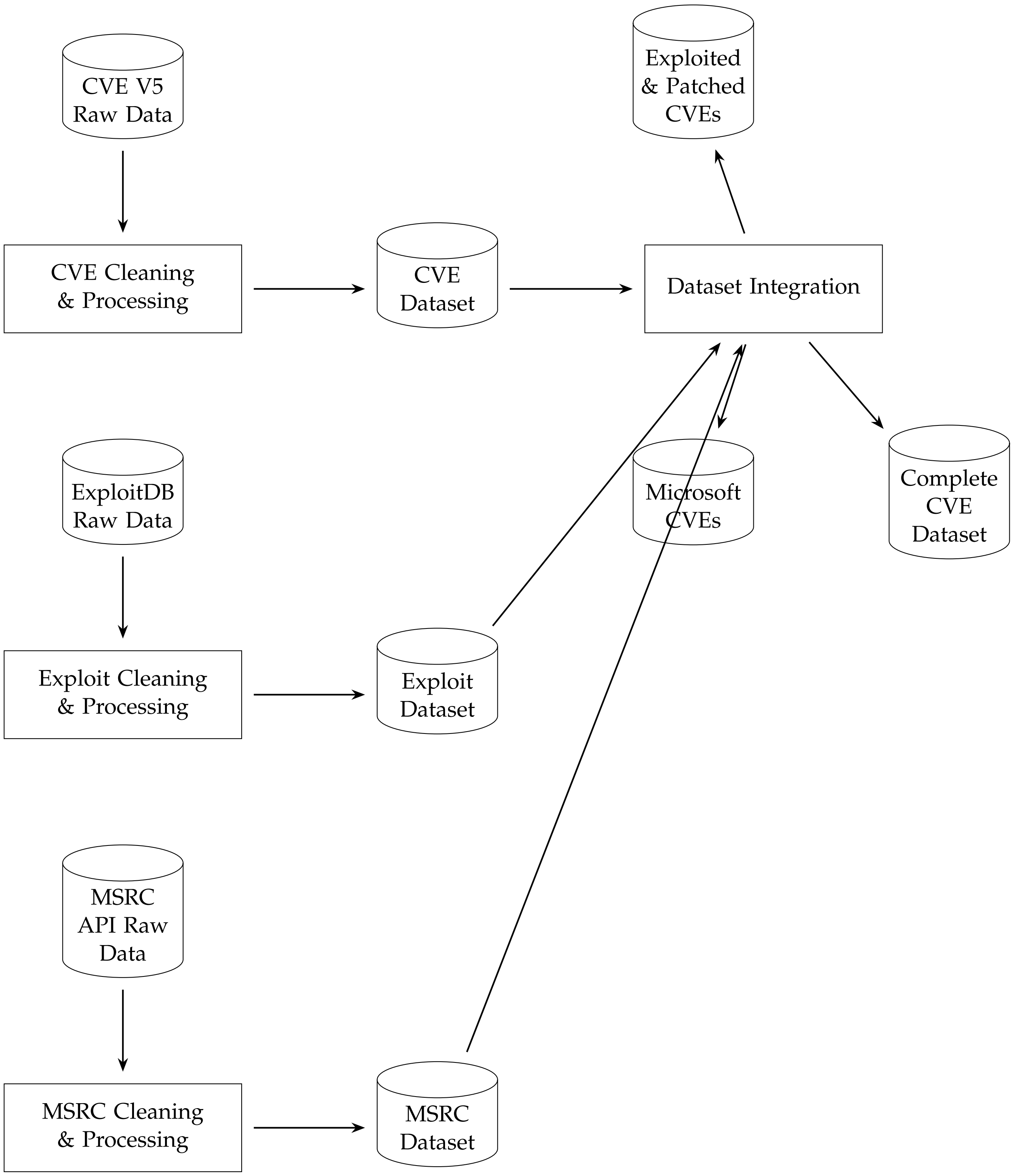


FIGURE 3.2: Data Preparation and Integration Process

The resulting integrated datasets provide a comprehensive foundation for analyzing vulnerability lifecycles, enabling investigation of the relationships between vulnerability characteristics, exploitation patterns, and patching timelines. The creation of specialized subsets, particularly the Exploited and Patched CVE dataset, supports focused analysis of the complete vulnerability lifecycle, addressing gaps identified in previous research.

## 3.4 Analytical Framework

### 3.4.1 Descriptive Analysis

The research began with a comprehensive descriptive analysis of the vulnerability landscape, exploring:

- **Temporal Distributions** analyzed the frequency and trends of vulnerabilities over time. This analysis revealed significant peaks in CVE disclosures during 2017-2021 and 2022-2024, with a remarkable 78% increase observed during the pandemic period (2020-2021). This accelerating trend aligns with findings from Ruohonen [130], who observed increasing temporal variations in vulnerability reporting, though our analysis reveals a more dramatic increase than previously documented.

- **Severity Distributions** examined the distribution of vulnerabilities across different severity levels. Our findings show a predominance of medium-severity vulnerabilities, challenging the notion of "severity inflation" described by some researchers [162], while supporting Cyentia's conclusion [34] that apparent increases in severity scores are more attributable to changes in scoring methodology than to fundamental changes in vulnerability characteristics.

- **Vulnerability Types** analyzed Common Weakness Enumerations (CWEs) associated with vulnerabilities to identify prevalent weaknesses. The analysis of 175,250 CVE-CWE pairs revealed that CWE-79 (Cross-site Scripting) remains the most prevalent weakness, accounting for 15.35% of all mapped weaknesses. This persistence aligns with Meunier's [102] observation that vulnerabilities often exist simultaneously at multiple abstraction levels, making them particularly resistant to comprehensive remediation.

- **Affected Platforms** investigated the distribution of vulnerabilities across different platforms, vendors, and products. Our analysis of approximately 2.5 million affected CPEs across 32,572 vendors revealed that Microsoft is among the top vendors with vulnerabilities, accounting for a significant portion of the vulnerability landscape. This finding aligns with Shahzad et al.'s [134] research on the distribution of vulnerabilities across major vendors.

- **Exploitation Patterns** examined the characteristics and trends of exploited vulnerabilities. Our analysis of 46,494 exploit records from ExploitDB revealed significant shifts in the temporal relationship between CVE reservations and exploit publications. The data demonstrates a dramatic evolution from predominantly pre-CVE exploits in early years (1999-2004, over 80%) to post-CVE exploits in recent years (2015-2024, over 80%), providing empirical support for Frei's temporal classification framework [46].

This descriptive analysis provided a foundation for understanding the broader context of vulnerability management and identifying areas for deeper investigation. The approach aligns with recommendations by Liao et al. [87] regarding the importance of comprehensive descriptive analysis in vulnerability research.

### 3.4.2 Time Series Analysis

Time series analysis was employed to examine temporal patterns in vulnerability disclosure, exploitation, and patching. The techniques used included:

- **Trend Analysis** identified long-term trends in vulnerability discoveries, exploit publications, and patch releases. This analysis revealed consistent growth in the number of vulnerabilities reported annually, with particular acceleration during the pandemic period. For Microsoft specifically, we observed a substantial increase in patching activity from 386 patched CVEs in 2016 to a peak of 2,126 patched CVEs in 2022, representing a 451% increase. This finding, presented in Chapter 5, highlights the growing challenges of vulnerability management in increasingly complex software environments.
- **Temporal Aggregation** analyzed data at different temporal granularities (daily, monthly, yearly) to reveal patterns at various scales. This approach enabled identification of both short-term fluctuations and long-term trends in vulnerability management processes. As demonstrated in Chapter 4's analysis of CWE distribution over time (Figure 4.6), this multi-scale temporal analysis revealed important patterns in the evolution of specific weakness types, such as the persistent growth of Cross-site Scripting (CWE-79) vulnerabilities.
- **Event Correlation** examined relationships between key events in the vulnerability lifecycle (e.g., correlation between patch releases and exploit publications). As detailed in Chapter 5, this analysis revealed that exploits tend to be published approximately one month before patches become available, with a mean difference of -31 days for Microsoft products. This finding extends research by Woo et al. [161] on the challenges of timely patch development.
- **Change Point Detection** identified significant shifts in vulnerability patterns over time. This analysis revealed notable changes in the relationship between exploit publication and CVE reservation, with a marked transition from predominantly pre-CVE exploits before 2012 to predominantly post-CVE exploits after 2012. As shown in Chapter 5's analysis of CVE-exploit race patterns (Figure 5.3), this shift suggests substantial improvements in vulnerability management practices, particularly in coordination between researchers, vendors, and security professionals.

These analyses helped reveal the evolution of vulnerability management practices and the changing dynamics between attackers and defenders. The approach aligns with recommendations by Rajasooriya et al. [123] regarding the importance of temporal modeling in understanding vulnerability lifecycles.

#### 3.4.2.1 Vulnerability Lifecycle Analysis

A core component of the research was the detailed analysis of vulnerability lifecycles, focusing on the timing and sequence of key events:

- **Time-to-Exploit Analysis** calculated and analyzed the time between vulnerability discovery (CVE reservation) and the first public exploit. Our analysis revealed a mean difference of 168 days between CVE reservation and exploit publication for Microsoft products, with significant variation across different severity levels. Notably, we found that critical vulnerabilities exhibited longer exploitation timelines (median 192 days) than medium-severity ones (median 138 days), challenging conventional assumptions in risk assessment frameworks. This detailed analysis, presented in Chapter 5, Section 5.3.1.1, provides crucial insights into the relationship between vulnerability severity and exploitation patterns.

- **Time-to-Patch Analysis** examined the time between vulnerability discovery and the release of patches. Our analysis revealed that Microsoft patches vulnerabilities based on severity, with critical vulnerabilities patched in a median of 54 days, compared to 106 days for low-severity vulnerabilities. This prioritization aligns with risk-based approaches to vulnerability management. However, as shown in Chapter 5's three-way comparison of patching times (Figure 5.16), exploited vulnerabilities exhibit substantially longer patching timelines, with critical exploited vulnerabilities requiring a median of 186 days to patch.

- **Exploit-to-Patch Gap Analysis** investigated the critical window between exploit availability and patch release. Our findings indicate that exploits tend to be published before patches become available, with a median gap of -8 days for Microsoft products. This negative median difference indicates that attackers frequently gain an advantage in the "race" between exploitation and patching. Chapter 5's exploitation-patching relationship analysis (Section 5.3.1.2) provides a detailed examination of this critical security window.

- **Lifecycle Patterns by Severity** analyzed how lifecycle timelines vary by vulnerability severity. Our analysis revealed counterintuitive patterns, with critical vulnerabilities taking longer to exploit (median 192 days) than medium-severity ones (median 138 days). This finding challenges assumptions in existing vulnerability prioritization frameworks and extends research by Costa et al. [31] on the limitations of current severity scoring systems.

- **Outlier Investigation** identified and analyzed cases with unusually long or short lifecycle timelines. Our analysis uncovered 24 CVEs (6% of the exploited and patched dataset) related to third-party software working on Microsoft platforms, some with extreme delays between exploitation and patching exceeding 1,000 days. As detailed in Chapter 5, Section 5.4.1, these outliers exemplify the "long tail" of vulnerability remediation documented by Allodi [8].

This lifecycle analysis built upon the framework established by Frei [46], extending it with a specific focus on Microsoft-related vulnerabilities and contemporary data. The approach aligns with recommendations by Franca et al. [45] regarding the need for structured frameworks that address the complete vulnerability lifecycle.

### 3.4.3 Comparative Analysis

Comparative analyses were conducted to identify patterns and relationships across different dimensions of the dataset:

- **Microsoft vs. General CVEs** compared the characteristics and lifecycles of Microsoft-specific vulnerabilities with the broader vulnerability landscape. Our analysis revealed that Microsoft's vulnerability landscape has transformed from 98.5% pre-CVE exploits in 1999-2000 to almost exclusively post-CVE exploits after 2015. This pattern aligns with findings from Shahzad et al. [134] on Microsoft's increasingly structured approach to vulnerability management.

- **Verified vs. Unverified Exploits** analyzed differences in patterns between verified and unverified exploits. Our analysis found that out of 22,130 unique CVEs with exploits, 19,392 (87.63%) have a single exploit, while 2,738 (12.37%) have multiple exploits. As shown in Chapter 4's analysis of exploit distribution (Figure 4.16), this distribution supports Allodi's [8] observations on the heavy-tailed nature of exploitation patterns.

- **CWE-Based Comparisons** examined how vulnerability lifecycles vary across different types of weaknesses. Our analysis of the top 10 most common CWEs revealed that CWE-119 (Improper Restriction of Operations within the Bounds of a Memory Buffer) and CWE-79 (Cross-site Scripting) have been consistently associated with high CVSS scores, indicating that they pose serious risks to software security. Chapter 4's detailed analysis of CWE patterns (Figure 4.6) provides insights into the evolution of these weakness types over time.

- **Product-Based Comparisons** investigated variations in vulnerability patterns across different Microsoft products. Our analysis revealed that Microsoft Office leads with 6,342 patched vulnerabilities, significantly ahead of other product families. Windows and Microsoft .NET follow with 4,869 and 4,867 patched CVEs respectively. This distribution reflects both the complexity and ubiquity of these products in enterprise environments. As shown in Chapter 4's product family analysis (Figure 4.25), this finding challenges common assumptions about vulnerability concentration in operating systems.

- **Microsoft-Specific vs. Third-Party Vulnerabilities** compared lifecycle patterns between vulnerabilities in Microsoft's own products and those in third-party software operating on Microsoft platforms. This analysis, presented in Chapter 5, Section 5.3.1, revealed dramatic differences, with 37.5% of third-party vulnerabilities showing extreme delays exceeding 1,000 days, compared to just 0.26% of Microsoft-specific products. This finding highlights significant challenges in cross-vendor vulnerability management and coordination.

- **CWE Distribution Comparison** analyzed differences between CWE distributions in Microsoft products versus the broader vulnerability landscape. As detailed in Chapter 4 (Table 4.11), Microsoft places significantly higher emphasis on memory safety issues (58.1% of top patched vulnerabilities) compared to the general vulnerability landscape (32.1%), while showing less focus on web vulnerabilities (7.4% versus 38.1%). This distinct profile reflects Microsoft's specific security challenges and priorities.

These comparative analyses provided deeper insights into factors influencing vulnerability lifecycles and helped identify specific patterns within the Microsoft ecosystem. The approach aligns with recommendations by Tenable [150] regarding the importance of vendor-specific vulnerability analysis for accurate risk assessment.

## 3.5 Implementation Environment

### 3.5.1 Technical Infrastructure

The research was conducted using a virtualized environment hosted on Amazon Web Services (AWS), consisting of:

- An EC2 instance running Ubuntu 22.04 LTS
- 8 vCPUs and 32GB RAM
- 1TB of attached storage

This infrastructure provided the necessary computational resources for data collection, processing, and analysis while ensuring scalability and reliability. The selection of AWS aligns with current best practices for cloud-based research environments, providing the flexibility and computational capacity required for large-scale data analysis.

### 3.5.2 Tools and Libraries

The research leveraged various open-source tools and libraries:

**Programming Language**: Python 3.10 served as the primary programming language, chosen for its robust ecosystem of data science and analysis libraries. Python's flexibility and extensive support for data manipulation, statistical analysis, and visualization made it an ideal choice for this research. **Data Processing:**

- Pandas 1.5.2 for data manipulation and analysis, providing powerful data structures and operations for working with structured data.
- NumPy 1.23.5 for numerical operations, supporting mathematical computations and array-based data processing.
- JSON for processing structured data, particularly for parsing the complex nested structures in CVE V5 data.

**Data Visualization:**

- Matplotlib 3.6.2 for basic plots and visualizations, enabling creation of static, animated, and interactive visualizations
- Seaborn 0.12.1 for statistical visualizations, providing a high-level interface for drawing attractive and informative statistical graphics

**Statistical Analysis:**

- SciPy 1.9.3 for scientific computing and statistical tests, supporting advanced statistical analysis of vulnerability data
- StatsModels 0.13.5 for statistical models, enabling more sophisticated modeling of vulnerability lifecycles

**Database Interaction:**

- SQLAlchemy 1.4.44 for database ORM, providing a SQL toolkit and Object-Relational Mapping system
- PyMySQL 1.0.2 for MySQL database connectivity, enabling efficient interaction with the research database

**API Interaction:**

Requests 2.28.1 for HTTP requests, supporting retrieval of data from the MSRC API and other web resources.

These tools provided a robust foundation for implementing the research methodology and conducting the required analyses. The selection of open-source tools aligns with current best practices in research methodology, ensuring reproducibility and transparency.

### 3.5.3 Database Implementation

A MySQL 8.0 database was used to store and manage the research data. The database schema was designed to capture the complex relationships between vulnerabilities, exploits, and patches, with the following key tables:

- **cvev5**: Storing comprehensive information about CVEs
- **exploit**: Containing details about known exploits
- **msrc_vuln**: Storing information about Microsoft security updates
- **msrc_threat**: Capturing threat information associated with vulnerabilities
- **msrc_remediation**: Containing details about remediation options
- **msrc_product**: Storing information about affected Microsoft products
- **cwe**: Containing Common Weakness Enumeration data
- **cve_exploited_patched**: A specialized table for lifecycle analysis

The database implementation facilitated efficient querying and analysis of the data, enabling both exploratory investigations and targeted research into specific aspects of vulnerability lifecycles.

## 3.6 Validation and Quality Assurance

### 3.6.1 Data Validation Techniques

To ensure the accuracy and reliability of the research data, several validation techniques were employed:

- **Cross-Source Validation**: Comparing data across multiple sources to identify and resolve inconsistencies. This approach was particularly important for validating exploitation data, as highlighted in Chapter 4's analysis of discrepancies between Microsoft-reported exploited CVEs (124) and Exploit DB records (468). This validation revealed significant differences in exploitation reporting that informed our methodological decisions.
- **Schema Validation**: Ensuring that all data conforms to predefined schemas and constraints. This validation was crucial for maintaining consistency across the diverse data sources integrated in this research. As shown in Chapter 4's analysis of CPE structures, strict schema validation ensured accurate parsing of complex platform identifiers.
- **Temporal Consistency Checks**: Verifying the logical consistency of date sequences (e.g., ensuring that patch dates do not precede CVE reservation dates, however, it will be mentioned ). These checks were essential for the lifecycle analyses presented in Chapter 5, which depended on accurate temporal relationships between vulnerability events.

**Note:** The data has been analysed in its original form, without any modifications, with the analysis focused solely on the insights derived directly from the data itself. *Inconsistency is mentioned*

- **Statistical Validation**: Using statistical methods to identify and investigate outliers and anomalous patterns. This approach informed Chapter 5's detailed analysis of lifecycle outliers, enabling identification of the 9 third-party component CVEs with extreme patching delays exceeding 1,000 days.
- **Data Completeness Assessment**: Evaluating the coverage of key attributes across different datasets. This assessment revealed significant differences in data completeness between sources, such as the finding that 4,607 CVEs (46.2%) from the MSRC patched dataset had no associated CWE information. As documented in Chapter 4's methodological workflow for CWE distribution analysis, this assessment informed decisions about which data sources to prioritize for different analyses.

These validation techniques helped maintain data quality throughout the research process.

### 3.6.2 Error Handling and Edge Cases

The research methodology incorporated robust error handling and edge case management:

- **Missing Data Handling**: Implementing appropriate strategies for handling missing data, including imputation where reasonable and explicit exclusion where necessary. As demonstrated in Chapter 4's product family analysis, this approach led to the methodological decision to use MSRC product information rather than CPE data from the CVE-V5 database, as the latter had missing information for 30.6% of patched CVEs.
- **Outlier Detection and Management**: Identifying statistical outliers and implementing appropriate strategies for analysis and interpretation. This approach was central to Chapter 5's detailed investigation of lifecycle outliers in Section 5.4, which revealed dramatic differences between Microsoft-specific products and third-party components.
- **Logging and Auditing**: Maintaining comprehensive logs of data processing activities to track and address potential issues. This practice ensured traceability of all data transformations, enabling verification of analytical results.
- **Many-to-Many Relationship Handling**: Developing specific processing workflows to handle complex relationships between entities. As illustrated in Chapter 4's product family analysis workflow (Figure 4.24,4.20) and Chapter 5's CWE distribution analysis workflow (Figure 4.26), this approach ensured accurate representation of vulnerabilities affecting multiple products or associated with multiple weakness types.

These measures ensured that the analysis remained robust despite the challenges inherent in working with complex, real-world data.

### 3.6.3 Consistency Checks

Regular consistency checks were performed to maintain the integrity of the research:

- **Internal Consistency**: Ensuring logical consistency within the dataset (e.g., report what data is telling us, checking that counts and relationships remained consistent after processing). These checks were particularly important for the integrated analyses presented in Chapter 5, which combined data from multiple sources to examine complete vulnerability lifecycles.
- **External Consistency**: Validating findings against external benchmarks and published literature. As noted throughout Chapters 4 and 5, our findings were compared with previous research by Frei [46], Allodi [8], Shahzad et al. [134], and others to identify areas of alignment and extension.
- **Reproducibility Testing**: Verifying that analytical processes produced consistent results when repeated. This testing was especially important for the complex data processing workflows described in Chapter 4, such as the product family processing workflow (Figure 4.24) and CWE distribution analysis workflow (Figure 4.26).
- **Cross-Metric Validation**: Comparing related metrics to ensure consistent patterns. As demonstrated in Chapter 5's three-way comparison of patching times (Figure 5.16), this approach revealed important insights into the relationship between exploitation status and patching efficiency across severity levels.

These checks helped ensure the reliability and validity of the research findings.

## 3.7 Limitations and Ethical Considerations

### 3.7.1 Dataset Limitations

The research acknowledges several limitations of the datasets used:

- **Incomplete Coverage**: The datasets may not capture all vulnerabilities, particularly those that remain undisclosed or are handled through private channels. As revealed in Chapter 4's analysis of exploitation reporting, significant discrepancies exist between Microsoft-reported exploited CVEs (124) and those listed in Exploit DB (468), suggesting incomplete exploitation reporting in both sources.
- **Reporting Biases**: The data is subject to reporting biases, with certain types of vulnerabilities potentially being over- or under-represented. Chapter 4's analysis of CWE distribution revealed that while Cross-site Scripting (CWE-79) is the most prevalent weakness overall, it appears only sixth in Microsoft's patching priorities, highlighting potential differences in reporting or prioritization across different contexts.
- **Temporal Scope**: The detailed patch analysis focuses primarily on vulnerabilities from 2016 onwards, limiting historical comparisons. While the broader exploit analysis in Chapter 4 extends back to 1999, the comprehensive three-point lifecycle analysis in Chapter 5 is limited to more recent vulnerabilities.

- **Vendor Specificity**: The patch information is specific to Microsoft, limiting the generalizability of some findings to other vendors. As shown in Chapter 5's analysis of exploitation-patching relationships, Microsoft's specific security practices may not reflect industry-wide patterns.
- **Product Information Inconsistencies**: As documented in Chapter 4's product family analysis workflow (Figure 4.16), 30.6% of patched CVEs lacked associated CPE information in the CVE-V5 database, necessitating the use of MSRC product information for comprehensive product family analysis. This limitation highlights the challenges of integrating data across different sources.
- **CWE Coverage Gaps**: Chapter 4's CWE distribution analysis workflow (Figure 4.26) revealed that 46.2% of CVEs from the MSRC patched dataset had no associated CWE information. This significant gap in weakness classification limits the comprehensiveness of vulnerability type analysis.

Important Note on Data Completeness and Currency

During our data collection and analysis process, we observed several limitations related to the completeness and currency of vulnerability information in public databases:
**CPE Data Incompleteness:** Of the 3,054 Microsoft-related CVEs that lacked associated CPE information in our dataset, 116 had no CPE data at all at the time of collection (April 2024), with 7 of these having been rejected in the CVE-V5 database. This absence of CPE information significantly impacts the ability to associate vulnerabilities with specific products and versions.
**Dynamic Updates to Official Repositories:** The CVE-V5 database repository is continuously maintained with hourly updates that can include significant changes to CVE records. For example, Figure 3.1 shows changes made to CVE-2017-9206 on September 17, 2024 (after our data collection), where both the published and reserved dates were corrected from 2022 to 2017.

CODE LISTING 3.1: GitHub commit showing date corrections for CVE-2017-9206 made after our data collection

```
@@ -31,7 +31,7 @@
 }
 ],
 "providerMetadata": {
-     "dateUpdated": "2022-10-03T16:23:08",
+     "dateUpdated": "2017-05-23T03:56:00Z",
      "orgId": "8254265b-2729-46b6-b9e3-3dfca2d5bfca",
      "shortName": "mitre"
 },
@@ -148,9 +148,9 @@
 "assignerOrgId": "8254265b-2729-46b6-b9e3-3dfca2d5bfca",
 "assignerShortName": "mitre",
 "cveId": "CVE-2017-9206",
-     "datePublished": "2022-10-03T16:23:08",
-     "dateReserved": "2022-10-03T00:00:00",
-     "dateUpdated": "2024-08-05T17:02:43.661Z",
+     "datePublished": "2017-05-23T03:56:00Z",
+     "dateReserved": "2017-05-22T00:00:00Z",
+     "dateUpdated": "2024-09-17T02:57:30.668Z",
 "state": "PUBLISHED"
 },
 "dataType": "CVE_RECORD",
```

**New CVEs Without CPE Data:** Some newly published CVEs may initially lack CPE information, which is added later. For instance, CVE-2024-2174 (a Chrome vulnerability) had no CPE data when collected in April 2024, but was later updated with detailed CPE entries for Google Chrome and Fedora Linux.
**Rejected CVEs in Vendor Patches:** We also identified cases where rejected CVEs appear in vendor patch information. For example, CVE-2022-1480 was rejected with the note "This candidate was withdrawn by its CNA. Further investigation showed that it was not a security issue," yet it appears in Microsoft's patched CVE dataset.
These observations highlight the need for real-time data streaming and updated pipeline processes to improve data quality and completeness, as noted by Sun et al. [145]. Their research on vulnerability database inconsistencies emphasizes the importance of continuous monitoring and reconciliation of vulnerability information across multiple sources. For future research, implementing automated systems to detect and incorporate database updates would significantly enhance the accuracy and currency of vulnerability lifecycle analyses.

These limitations are acknowledged and considered in the interpretation of research findings.

### 3.7.2 Methodological Constraints

The research methodology has several constraints that should be considered:

- **Correlation vs. Causation**: The analysis identifies correlations between various factors, but establishing causation requires additional research. Chapter 5's findings on the relationship between vulnerability severity and exploitation timing, where critical vulnerabilities show longer exploitation timelines than medium-severity ones, highlight correlations that may have multiple possible causal explanations.
- **Selection Bias**: The focus on vulnerabilities with both exploits and patches may introduce selection bias, as these may differ from the broader vulnerability population. Chapter 5's three-way comparison of patching times (Figure 5.16) demonstrates significant differences between exploited and non-exploited vulnerabilities, suggesting that the subset of exploited vulnerabilities may have distinct characteristics.
- **Limited Contextual Information**: The quantitative approach may not capture all contextual factors influencing vulnerability lifecycles, such as organizational priorities or resource constraints. Chapter 5's analysis of outlier cases with extreme patching delays identified potential factors like cross-vendor coordination challenges, but comprehensive understanding of these contexts remains limited.
- **Analytical Trade-offs**: As demonstrated in Chapter 4's product family and CWE distribution analyses, methodological decisions about data sources involve trade-offs between coverage, specificity, and consistency. The decision to use MSRC product information rather than CPE data improves coverage but potentially limits comparability with broader vulnerability datasets.
- **Temporal Alignment Challenges**: The precise determination of event chronology can be challenging, particularly for older vulnerabilities or those with limited documentation. Chapter 5's lifecycle analysis acknowledges these challenges by using the earliest documented date for each lifecycle event as the reference point.

These constraints are addressed through careful interpretation and acknowledgment of the scope of the research findings.

### 3.7.3 Ethical Research Practices

The research was conducted in accordance with ethical principles for cybersecurity research:

- **Responsible Use of Data**: All data used in this research was obtained from public sources and used in accordance with relevant terms of service. The research relied on established vulnerability databases and vendor security advisories rather than active vulnerability discovery or exploitation.

- **No Exploitation**: The research did not involve actual exploitation of vulnerabilities or any activities that could compromise system security. Analysis focused on previously documented exploits and patches rather than developing new exploitation techniques.
- **Disclosure Sensitivity**: Care was taken to present findings in a manner that does not facilitate malicious activities or expose unpatched vulnerabilities. Chapter 5's detailed analysis of lifecycle patterns and exploitation timelines focuses on general patterns rather than specific exploitation techniques or unpatched vulnerabilities.
- **Privacy Consideration**: The research focused exclusively on technical vulnerability data without involving personal data or identifiable user information. All analysis was conducted on anonymized vulnerability metrics without reference to specific security incidents affecting individuals.

These ethical considerations guided all aspects of the research process.

## 3.8 Summary

This chapter has outlined the comprehensive methodology employed to investigate the lifecycle of Common Vulnerabilities and Exposures (CVEs), with a specific focus on those affecting Microsoft products. The research design combines quantitative analysis of large-scale vulnerability datasets with qualitative interpretation, grounded in established theoretical frameworks.

The methodology leverages multiple authoritative data sources, including the CVE V5 database, ExploitDB, and the Microsoft Security Response Center API, to ensure comprehensive coverage and enable cross-validation. Rigorous data processing and integration techniques were applied to create a cohesive analytical dataset, with specialized subsets for focused analysis.

The analytical framework encompasses descriptive analysis, time series analysis, vulnerability lifecycle analysis, and comparative analysis, providing a multi-faceted approach to understanding vulnerability dynamics. The research implementation used sophisticated processing workflows for specific analyses, such as the product family analysis and CWE distribution analysis detailed in Chapters 4 and 5. These workflows employed hierarchical categorization, many-to-many relationship handling, and strategic data source selection to overcome challenges with incomplete or inconsistent data.

Particular attention was given to temporal metrics and statistical approaches for analyzing vulnerability lifecycles. The research calculated key metrics such as Time to Exploit, Time to Patch, and Exploit-Patch Gap, using multiple statistical indicators (mean, median, mode, standard deviation) to provide a nuanced understanding of these temporal relationships. This approach extends beyond previous studies that often relied solely on averages, enabling identification of both typical patterns and significant outliers.

The methodology acknowledges and addresses several important data challenges, including the significant discrepancies between exploitation data reported by Microsoft and that available in the Exploit Database, the absence of CWE information for 46.2% of patched CVEs, and the lack of CPE information for 30.6% of Microsoft patched vulnerabilities. These challenges were addressed through strategic data source selection, cross-validation across multiple sources, and careful interpretation of findings.

The research was implemented using a robust technical infrastructure and open-source tools, with careful attention to data validation and quality assurance. While acknowledging limitations and constraints, this methodology provides a solid foundation for investigating the complex dynamics of vulnerability lifecycles and generating valuable insights for improving security practices.

The methodological approach detailed in this chapter has enabled the comprehensive analyses presented in Chapters 4 and 5, revealing important patterns in vulnerability characteristics, exploitation dynamics, and patching practices. These findings extend our understanding of vulnerability lifecycles while providing empirical evidence to support more effective security strategies and vulnerability management processes.

# CHAPTER 4

# Vulnerability, Exploit and Patch Analysis

In this chapter, we perform Exploratory Data Analysis (EDA) focusing on Common Vulnerabilities and Exposures (CVEs) and their associated exploits. The analysis aims to uncover trends, distributions, and characteristics of vulnerabilities reported over time. By examining the CVE dataset, which includes over 246,000 entries, we seek to understand the patterns in vulnerability disclosures, severity levels, and associated weaknesses. This investigation is enhanced by analyzing the relationships between CVEs and Common Weakness Enumerations (CWEs), and exploring the context of exploits related to these vulnerabilities. These insights are pivotal for improving cybersecurity practices and addressing prevalent security issues.

The systematic collection and analysis of vulnerability-related Open-Source Intelligence (OSINT) has transformed security research by providing rich, publicly accessible datasets that enable deeper understanding of vulnerability characteristics and distributions [131]. As Mittal et al. [109] noted, OSINT encompasses a wide range of information sources, from technical databases to social media discussions, providing researchers and security professionals with comprehensive insights without requiring privileged access.

Building upon established research frameworks, this chapter examines multiple dimensions of vulnerabilities and exploits:

- Temporal distributions of vulnerabilities to identify trends and patterns over time
- Severity distributions to understand the impact spectrum of reported vulnerabilities
- Common Weakness Enumeration (CWE) analysis to categorize and contextualize vulnerability types
- Exploit characteristics and their relationships with CVEs to assess real-world exploitation patterns
- Platform and vendor analysis to identify the most affected systems and providers

This holistic approach aligns with the structured methodologies advocated by researchers such as Pastor et al. [117], who emphasized the importance of integrating multiple data sources for comprehensive vulnerability intelligence. By analyzing both vulnerabilities and their associated exploits, we gain insights into not only the theoretical security weaknesses but also their practical implications in real-world attack scenarios.

The findings from this exploratory analysis provide a foundation for the more detailed temporal lifecycle analysis presented in Chapter 5, where we examine the relationships between vulnerability disclosure, exploitation, and patching over time. The current chapter focuses on the static characteristics and distributions of vulnerabilities and exploits, establishing the context necessary for understanding their dynamic interactions throughout their lifecycles.

## 4.1 CVE Analysis

The analysis of Common Vulnerabilities and Exposures (CVEs) is crucial for understanding the landscape of cybersecurity threats. This section examines the characteristics, distributions, and trends of CVEs, providing insights into the evolution of security vulnerabilities over time.

### 4.1.1 Overview of CVE Data

The *cvev5* table contains a comprehensive list of CVEs, with a total of 246,422 entries up to April 2024. Each entry includes information such as the CVE ID, description, publication date, severity scores, and associated weaknesses (CWEs). This dataset represents a pivotal resource for analyzing the nature and evolution of vulnerabilities across the cybersecurity landscape.

### 4.1.2 Distribution of CVEs Over Time

Analyzing the distribution of CVEs over time provides insights into trends in vulnerability disclosures. Figure 4.1 shows the number of CVEs reported annually from 1999 to 2024.

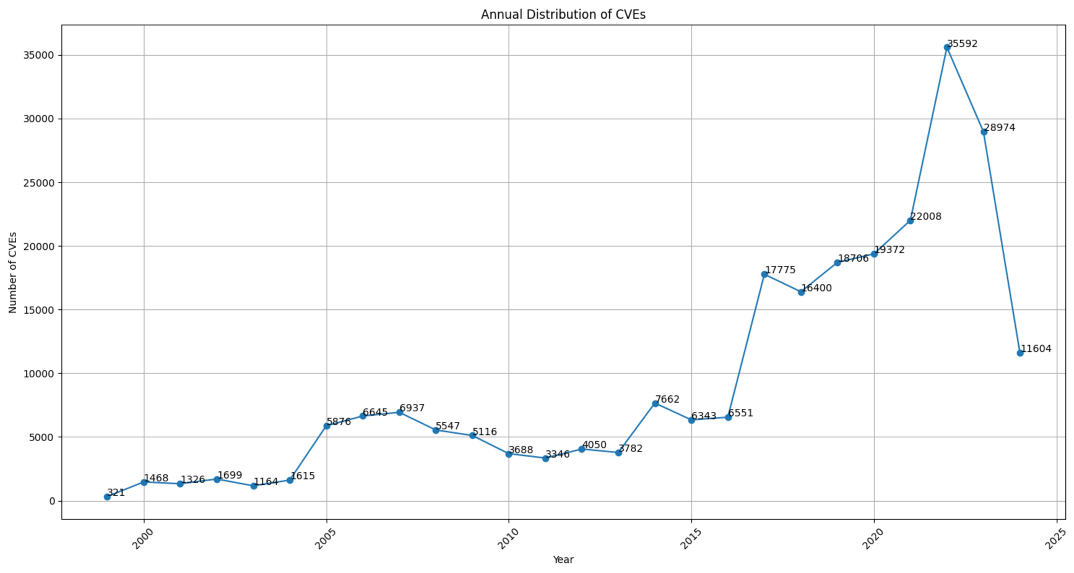


FIGURE 4.1: Annual distribution of CVEs from 1999 to 2024.

The data indicates significant peaks in CVE disclosures during the periods of 2017-2021 and 2022-2024. These peaks could be attributed to increased awareness and reporting of vulnerabilities, as well as advancements in detection techniques. A notable 78% increase in CVEs was observed during the pandemic (2020-2021), attributed to rapid digital transformation, vulnerabilities in remote work infrastructure, and an increased focus on security research [83].

According to a report conducted by Cyentia Institute 2023 [34], the number of CVEs has been steadily increasing since the introduction of the CVE Numbering Authorities (CNAs) process in early 2017, leading to a consistent rise in weekly CVE publications by about 10% per year. By 2025, it is expected that an average of 547 new CVEs will be reported each week, with some weeks seeing as many as 1,250 new CVEs.

This accelerating trend in CVE disclosures aligns with findings from Ruohonen [130], who observed increasing temporal variations in vulnerability reporting. However, our analysis reveals a more dramatic increase than previously documented, particularly during the pandemic period (2020-2021). The 78% increase observed during this period extends beyond the consistent 10% annual growth predicted by Cyentia Institute [34], suggesting that extraordinary circumstances like global pandemic conditions may introduce additional factors affecting vulnerability discovery and reporting rates that were not captured in previous trend analyses.

### 4.1.3 Severity of CVEs

The severity of CVEs is typically measured using the Common Vulnerability Scoring System (CVSS). The CVSS provides a standardized way to capture the principal characteristics of a vulnerability and produce a numerical score reflecting its severity. Table 4.1 presents the distribution of CVEs across different severity levels.

| Severity Level | Number of CVEs |
|---|---|
| Low | 34,817 |
| Medium | 125,331 |
| High | 68,892 |
| Critical | 17,382 |

TABLE 4.1: Distribution of CVEs by severity level.

The distribution of CVEs across different severity levels outlined in Figure 4.2 highlights the wide range of vulnerabilities that exist. Interestingly, while the CVSSv3 introduced a more refined scoring system, the average severity of CVEs has not increased over the last decade.

The predominance of medium-severity vulnerabilities in our dataset parallels observations by Shahzad et al. [134], who found that the majority of vulnerabilities tend to cluster in the middle of the severity spectrum rather than at extremes. However, unlike their findings which suggested a gradual increase in average severity over time, our data indicates stability in overall severity ratings despite the introduction of CVSSv3. This contradicts the notion of "severity inflation" observed by some researchers [162], while supporting Cyentia's conclusion [34] that apparent increases in severity scores are more attributable to changes in scoring methodology than to fundamental changes in vulnerability characteristics.

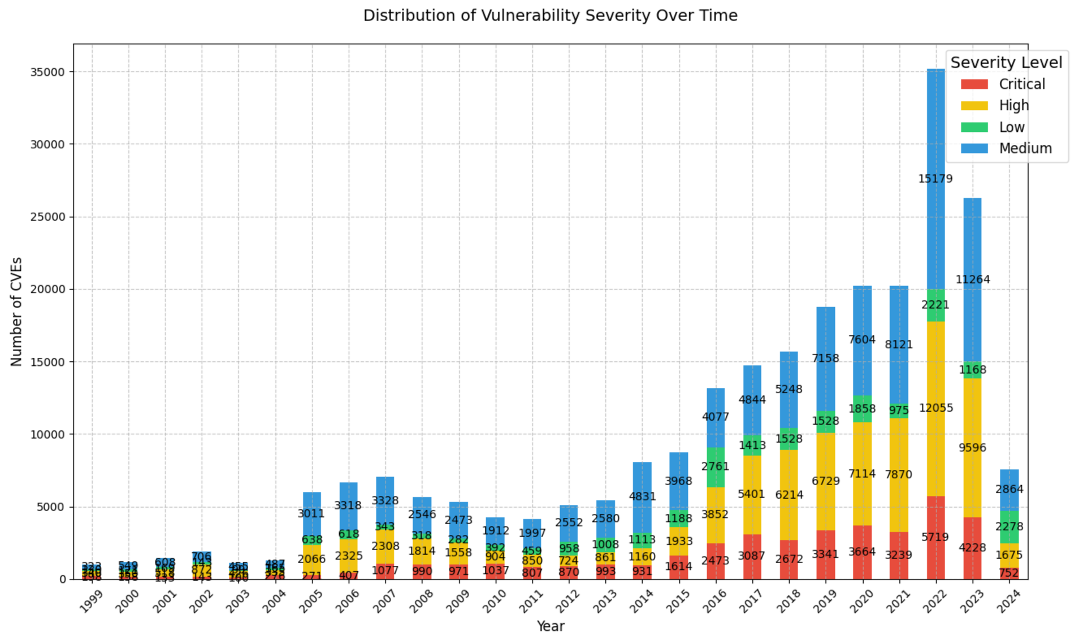


FIGURE 4.2: Annual distribution of CVE CVSS scores (1999-2024)

According to the recent report by Cyentia institute [34], although CVSSv3 scores tend to be higher on average compared to CVSSv2, this increase in score is more a result of changes in scoring criteria rather than an actual increase in vulnerability severity.

### 4.1.4 Common Platform Enumeration (CPE) and Vulnerability

Common Platform Enumeration (CPE) is a standardized naming scheme for IT systems, software, and packages. It uses a formal name format based on the Uniform Resource Identifier (URI) syntax. The CPE naming format includes several components, allowing for a structured way to identify and describe products and their versions.

#### 4.1.4.1 CPE Structure

A CPE Name (CPE WFN - Well-Formed Name) follows this structure:

```
cpe:cpe_version:part:vendor:product:version:update:architecture:language:modifier
```

Where:

- **part**: Defines the type of system and can be `a` for Application, `h` for Hardware, or `o` for Operating System.
- **vendor**: The name of the software developer.
- **product**: The name of the product.
- **version**: The version number of the product.
- **update**: Update or revision of the product.
- **edition**: Edition of the product.
- **language**: Language of the product (if applicable).

#### 4.1.4.2 Vulnerable System Types

CPE data reveals the following statistics for vulnerable systems:

- **CPE**: Number of CPEs *(Products)* affected is around 2.5 Million.
- **Vendor**: Number of distinct vendors is 32,572.
- **Product**: Number of distinct products is 146,783.
- **Applications (a)**: Over 1,110,415 vulnerable applications, potentially including multiple versions and instances of the same application.
- **Hardware (h)**: Over 461,952 vulnerable hardware items, including multiple versions of the same hardware.
- **Operating Systems (o)**: Over 899,967 vulnerable operating systems, which may include various versions of the same OS.

To better understand the distribution of vulnerabilities, we present two figures:

- **Figure 4.3**: Top 10 vulnerable products by the number of unique CVEs affecting the product or its versions. (`One CVE might affect multiple versions of the same product, but it is counted only once per product.`)
- **Figure 4.4**: Top 10 vulnerable vendors by the number of unique CVEs affecting their products. (`One CVE might affect multiple products or versions from the same vendor, but it is counted only once per vendor.`)

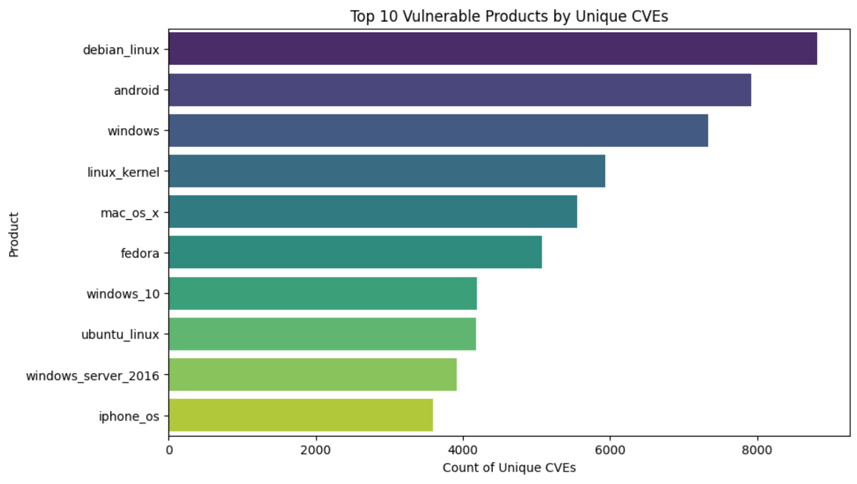


FIGURE 4.3: Top 10 Vulnerable Products

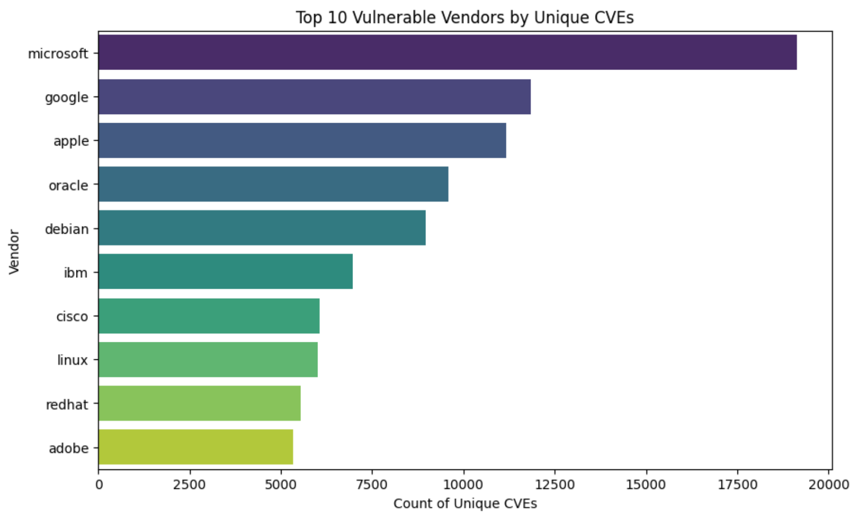


FIGURE 4.4: Top 10 Vulnerable Vendors

The figures highlight the distribution of vulnerabilities among products and vendors, offering insights into the most frequently affected components and manufacturers. These visualizations help in identifying trends and areas of focus for improving security practices and mitigating risks.

### 4.1.5 Vulnerabilities with Common Weakness Enumeration (CWE)

Each CVE can be associated with one or more weaknesses, categorized under the Common Weakness Enumeration (CWE) which is a standardized list of software and hardware weaknesses that have security implications. A "weakness" refers to a condition in software, firmware, hardware, or a service component that could lead to vulnerabilities under certain circumstances. Our dataset includes detailed information on CWEs associated with CVEs, which provides insights into the most prevalent and critical weaknesses.

#### 4.1.5.1 Dataset Overview

Our analysis of CWEs revealed the following:

- Number of unique CVEs: 246,422
- Number of CVEs with at least one associated CWE: 162,095
- Number of unique CWEs: 593
- Total number of weaknesses (CWE/CVE pairs): 175,250

#### 4.1.5.2 Distribution of CVE and CWE

Figure (4.5) illustrates the distribution of CVEs with and without associated CWEs over time. As shown in the plot, the number of CVEs with at least one CWE has generally increased over time, indicating a growing trend in associating vulnerabilities with specific weaknesses. However, a significant portion of CVEs still lack a CWE or have a CWE classified as ’NVD-CWE-noinfo’, ’NVD-CWE-Other’, or an empty string. These CVEs, represented by the ’Total No CWE / Specific CWE’ label in the visualization, lack a CWE due to limitations in the source NVD database [111].

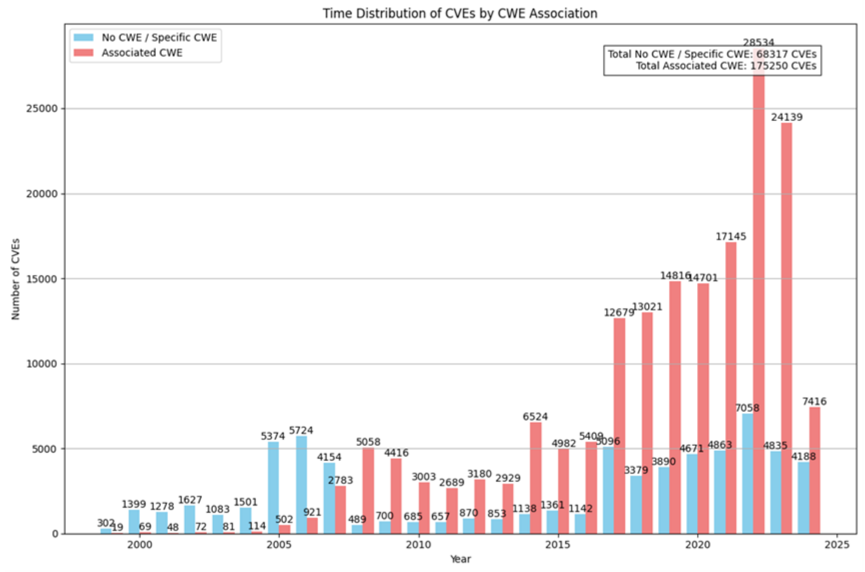


FIGURE 4.5: Distribution of CVEs with or without associated CWE.

### 4.1.5.3 Most Common CWEs in Vulnerabilities

The top 10 most common weaknesses in our dataset are listed in Table 4.2. These weaknesses represent the most frequently exploited types of vulnerabilities, indicating areas that require focused security efforts.

| CWE ID | Name | Number of affected CVE |
|---|---|---|
| CWE-79 | Cross-site Scripting (XSS) | 26,728 |
| CWE-119 | Memory Buffer Restriction | 11,906 |
| CWE-89 | SQL Injection | 11,356 |
| CWE-20 | Improper Input Validation | 10,446 |
| CWE-787 | Out-of-bounds Write | 10,183 |
| CWE-200 | Information Exposure | 7,852 |
| CWE-22 | Path Traversal | 5,902 |
| CWE-125 | Out-of-bounds Read | 5,866 |
| CWE-352 | Cross-Site Request Forgery (CSRF) | 5,630 |
| CWE-416 | Use After Free | 4,141 |

TABLE 4.2: Top 10 CWEs associated with CVEs.

### 4.1.5.4 Most Reoccurring CWEs Over Time

In the analysis of Common Weakness Enumerations (CWEs) within our datasets, certain vulnerabilities appear recurrently over time. The recurrence of Common Weakness Enumerations (CWEs) over time can be attributed to several factors, supported by data from our findings regarding the most prevalent CWEs as shown in Appendix E. The following Table 4.3 presents all top ten CWEs with their descriptions, total CVEs, trends, and key insights.

TABLE 4.3: Top 10 Common Weakness Enumerations by Occurrence

| CWE ID | Description | Total CVEs | Trend | Key Insight |
|---|---|---|---|---|
| CWE-79 | Cross-site Scripting (XSS) | 26,725 | Significant rise from 2007, peaked 2019 (4,800 CVEs), remained high through 2020 | Remains major security concern despite increased awareness |
| CWE-119 | Memory Buffer Bounds Restriction | 11,902 | Steady increase, major peak in 2017 (2,100 CVEs), decline after 2021 | Persistent in low-level languages (C/C++), affects both new and legacy systems |
| CWE-89 | SQL Injection | 11,356 | Sharp rise late 2000s, spikes in 2007-2008, major peak in 2021 (2,350 CVEs) | Critical vulnerability for older applications and poorly designed database interactions |
| CWE-20 | Improper Input Validation | 10,445 | Consistent presence, peak in 2021 (1,250 CVEs) | Many applications still failing to implement strong input validation |
| CWE-787 | Out-of-bounds Write | 10,181 | Notable increase from 2016, peaked 2021 (2,200 CVEs) | Memory safety remains significant concern, especially in performance-critical environments |
| CWE-200 | Exposure of Sensitive Information | 7,852 | Steady increase, peaks around 2017 (1,000 CVEs) | Reflects increasing emphasis on data privacy and security |
| CWE-22 | Path Traversal | 5,902 | Sharp rise 2007, peak in 2021 (1,000 CVEs) | Ongoing issues with file access control and user input handling |
| CWE-125 | Out-of-bounds Read | 5,864 | Significant rise from 2016, peaked 2021 (1,000 CVEs) | Vulnerabilities in array indexing and pointer arithmetic in C/C++ |
| CWE-352 | Cross-Site Request Forgery (CSRF) | 5,630 | Present mid-2000s, increased thereafter, peak in 2021 (1,050 CVEs) | Weaknesses in web session handling despite awareness |
| CWE-416 | Use After Free | 4,139 | Gained significance from 2011, peaked 2021 (800 CVEs) | Challenges in memory management, especially in systems programming |

**Root Causes for CWE Persistence according to CWE insights from Table 4.3**

1. **Legacy Systems and Codebases:** Continued use of outdated code not compliant with modern security practices (CWE-79: 15.35% of total CWEs)

2. **Inadequate Security Practices:** Insufficient validation and poor input handling (CWE-20: 6.00% of total)

3. **Complexity and Integration Issues:** Problems emerging from system integration (CWE-119: 6.84% of total)

4. **Lack of Awareness and Training:** Insufficient developer education on secure coding practices (CWE-89: 6.52% of total)

To visualize the distribution and occurrence of these top 10 CWEs over time, a plot is provided in Figure 4.6. This plot allows us to analyze the trend of these weaknesses and determine whether they are being addressed or continue to be significant issues. Notably, some CWEs consistently appear in an increasing number of CVEs over time, which supports our earlier observations about persistent vulnerabilities. This trend indicates that these weaknesses may not be effectively mitigated or that new vulnerabilities with similar characteristics are emerging.

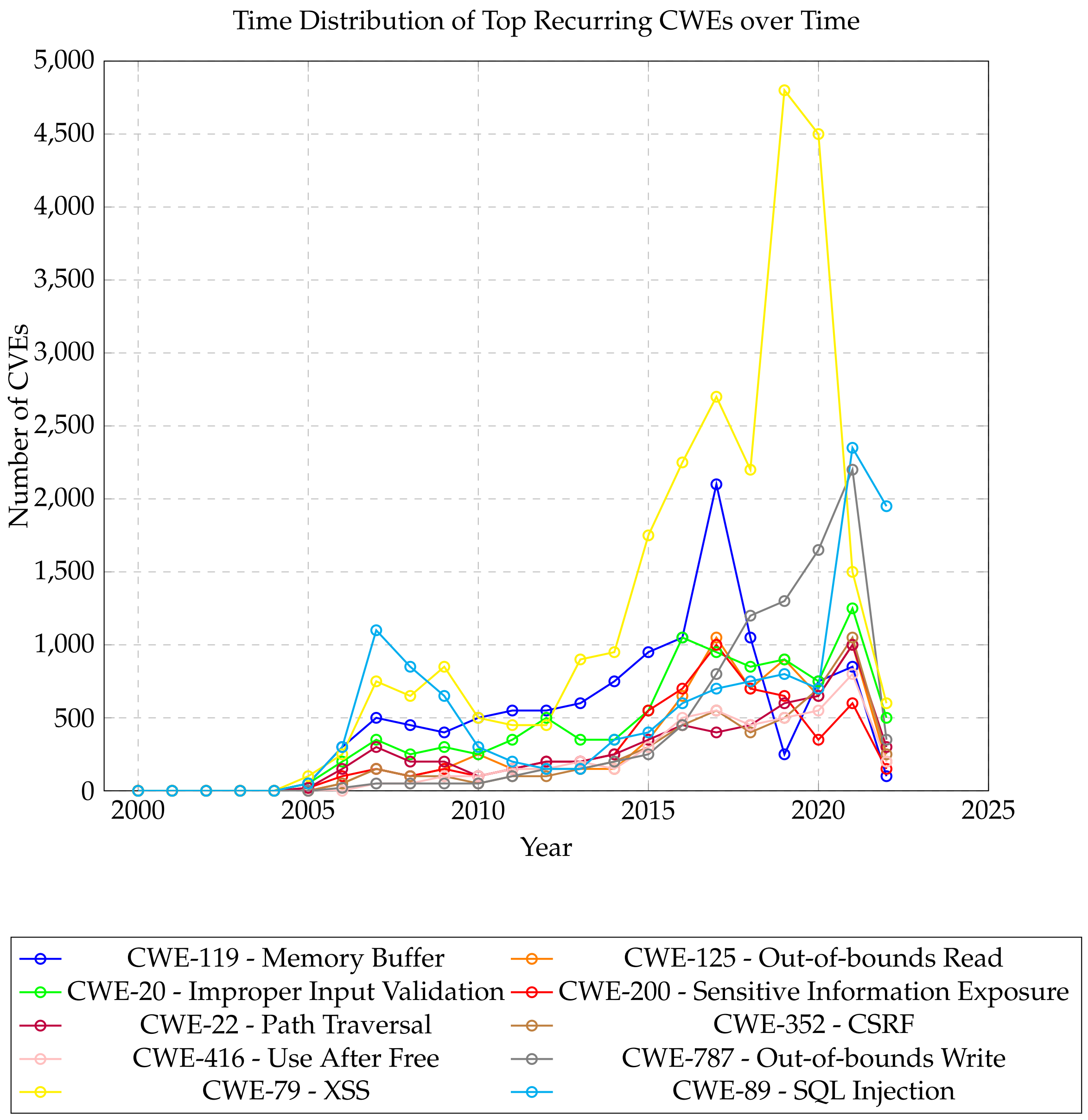


FIGURE 4.6: Temporal Distribution of Top 10 Recurring CWEs

#### 4.1.5.4.1 Key Observations from Temporal Analysis

1. **2019-2021 Vulnerability Peak:** Nearly all CWEs showed significant increases during this period, with CWE-79 (XSS) reaching almost 5,000 CVEs in 2019
2. **Collective Decline in 2022:** All vulnerabilities show a sharp decrease in 2022, which may indicate improved security practices or reporting changes
3. **Persistent Web Vulnerabilities:** XSS (CWE-79) and SQL Injection (CWE-89) remain among the most prevalent despite being well-documented for over two decades
4. **Growing Memory Safety Issues:** Out-of-bounds Write (CWE-787) shows consistent growth since 2016, suggesting increasing exploitation of memory safety vulnerabilities

5. **Cyclic Patterns:** Several vulnerabilities show cyclic patterns of increase and decrease, potentially tied to technology trends or security awareness campaigns

**Yearly Observations**

- **2008-2010:** Many CWEs saw a rise, particularly CWE-79 (XSS), CWE-89 (SQL Injection), and CWE-119 (Buffer Overflow). This period may coincide with increased awareness and reporting of web application vulnerabilities and advancements in detection tools.
- **2015-2020:** Most CWEs, including CWE-79, CWE-119, and CWE-787, experienced peaks. This period also aligns with the rise of cybersecurity awareness, bug bounty programs, and more advanced penetration testing techniques.
- **2021-2023:** While some CWEs (like CWE-89) decreased, others, such as CWE-79 (XSS) and CWE-787 (Out-of-bounds Write), remained high or increased, indicating ongoing issues with web application security and memory safety.

The table 4.4 represents the yearly observation for the top 10 recurring CWEs Data Summary.

| CWE ID | Total CVEs | Notable Years (High CVEs) |
|---|---|---|
| CWE-79 | 26,725 | 2018, 2021, 2022 |
| CWE-119 | 11,902 | 2008, 2017, 2022 |
| CWE-89 | 11,356 | 2009, 2010, 2022 |
| CWE-20 | 10,445 | 2014, 2017-2019 |
| CWE-787 | 10,181 | 2018-2020 |
| CWE-200 | 7,852 | 2017, 2019 |
| CWE-22 | 5,902 | 2007-2009 |
| CWE-125 | 5,864 | 2016, 2019-2022 |
| CWE-352 | 5,630 | 2013-2014, 2021 |
| CWE-416 | 4,139 | 2011, 2019-2020 |

TABLE 4.4: Summary of Top 10 Most Recurring CWEs Over the Years

The persistence of specific CWEs across multiple years reflects the "technical debt" phenomenon described by Akhoundali et al. [4], where foundational weaknesses continue to manifest despite awareness of their existence. Our findings extend the work of Meunier [102], who observed that vulnerabilities often exist simultaneously at multiple abstraction levels. This is particularly evident in the case of CWE-79 (Cross-Site Scripting), which has maintained its position as the most prevalent weakness type since 2007. The recurring nature of these vulnerabilities suggests that, as argued by Ozment and Schechter [114], many vulnerabilities may be introduced during initial development and remain undiscovered for years, challenging the assumption that code security improves significantly with age through natural discovery and remediation processes.

#### 4.1.5.5 Analysis of Top 10 Recurring CWEs for the last five years (2019-2023)

We compare the top 10 recurring Common Weakness Enumerations (CWEs) over the last 5 years (2019–2023) with the top 10 recurring CWEs for the entire dataset. This analysis will help us determine if there are significant changes in the vulnerability weaknesses affecting software over time.

#### 4.1.5.5.1 Top CWEs (2019–2023)

The below Figure 4.7 presents the top 10 CWEs in the period between 2019 and 2023, and the Table 4.5 including their occurrence count, percentage from the total CVEs, and the year they were first noticed.

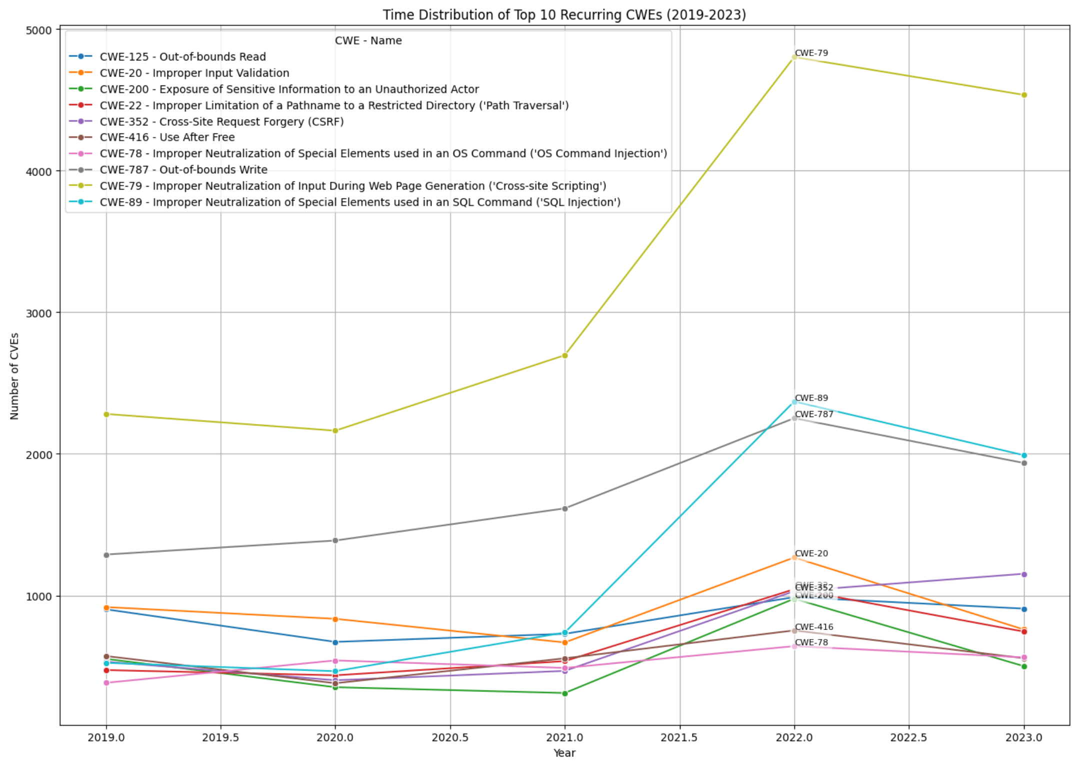


FIGURE 4.7: Distribution of the Top 10 Reoccurring CWEs Over Time for the period (2019–2023)

TABLE 4.5: Top 10 CWEs (2019–2023)

| CWE ID | Name | Count | Percentage (%) | First Noticed Year |
|---|---|---|---|---|
| CWE-79 | Cross-site Scripting (XSS) | 16,482 | 15.33 | 2019 |
| CWE-787 | Out-of-bounds Write | 8,480 | 7.88 | 2019 |
| CWE-89 | SQL Injection | 6,088 | 5.66 | 2019 |
| CWE-20 | Input Validation | 4,449 | 4.14 | 2019 |
| CWE-125 | Out-of-bounds Read | 4,198 | 3.90 | 2019 |
| CWE-352 | Cross-Site Request Forgery | 3,583 | 3.33 | 2019 |
| CWE-22 | Path Traversal | 3,234 | 3.01 | 2019 |
| CWE-416 | Use After Free | 2,818 | 2.62 | 2019 |
| CWE-200 | Exposure of Sensitive Info | 2,693 | 2.50 | 2019 |
| CWE-78 | OS Command Injection | 2,617 | 2.43 | 2019 |

#### 4.1.5.5.2 Top CWEs (All Years)

The table below summarizes the top 10 recurring CWEs across all years in the dataset.

TABLE 4.6: Top 10 CWEs (All Years)

| CWE ID | Name | Count | Percentage (%) |
|---|---|---|---|
| CWE-79 | Cross-site Scripting (XSS) | 26,728 | 15.35 |
| CWE-119 | Improper Restriction of Operations in Memory | 11,906 | 6.84 |
| CWE-89 | SQL Injection | 11,356 | 6.52 |
| CWE-20 | Input Validation | 10,446 | 6.00 |
| CWE-787 | Out-of-bounds Write | 10,183 | 5.85 |
| CWE-200 | Exposure of Sensitive Info | 7,852 | 4.51 |
| CWE-22 | Path Traversal | 5,902 | 3.39 |
| CWE-125 | Out-of-bounds Read | 5,866 | 3.37 |
| CWE-352 | Cross-Site Request Forgery | 5,630 | 3.23 |
| CWE-416 | Use After Free | 4,141 | 2.38 |

Upon comparing the two sets of top CWEs, several trends can be observed:

- **CWE-79 (Cross-Site Scripting)** continues to dominate both in the last 5 years and across all years, maintaining its position as the most recurring vulnerability.
- **CWE-119 (Improper Restriction of Operations in Memory)**, which ranks 2nd in the overall list, is notably absent from the top 10 for 2019–2023, indicating a potential reduction in this type of vulnerability in recent years.
- **CWE-787 (Out-of-bounds Write)** and **CWE-89 (SQL Injection)** remain in the top 5 across both time frames, suggesting persistent weaknesses in these areas.
- Newer vulnerabilities such as **CWE-78 (OS Command Injection)** appear in the 2019–2023 list but are not in the overall top 10, possibly due to more recent discovery and reporting trends.

#### 4.1.5.6 Reoccurring CWEs and its Related CVE CVSS Scores Over Time

Further analysis, as depicted in (Figure 4.8), explores whether the severity of CVEs affected by these CWEs has evolved over time.

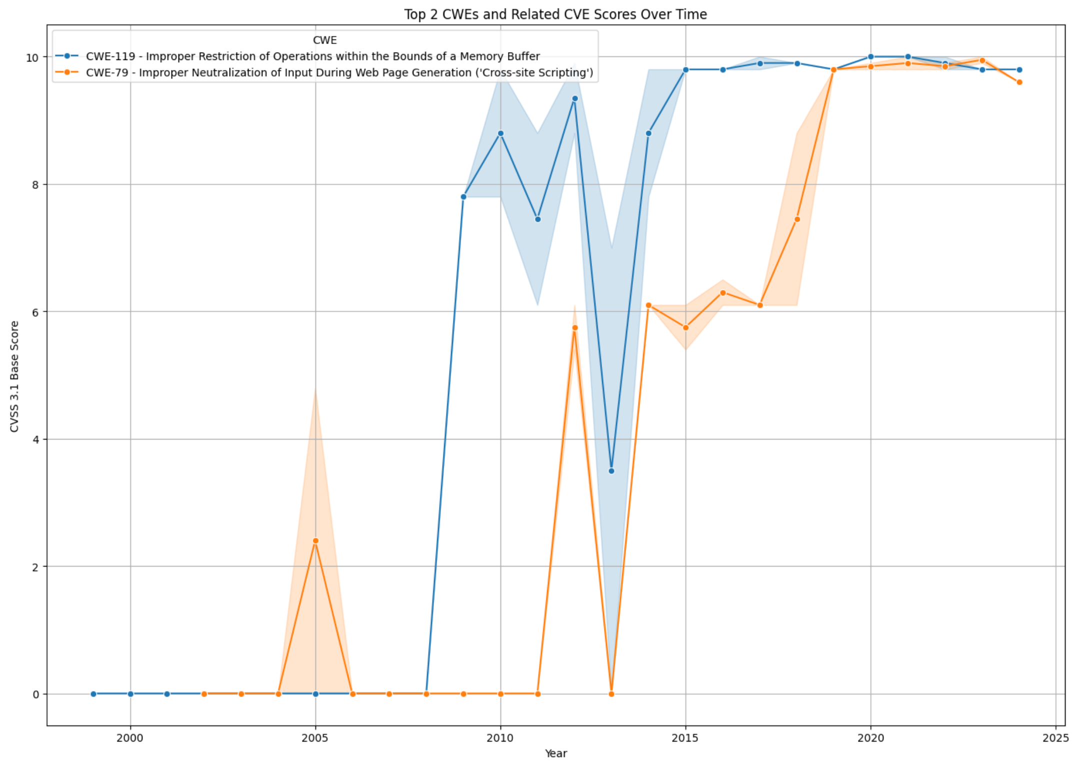


FIGURE 4.8: Top 2 CWEs and Related CVE Scores Over Time

This (Figure 4.8) illustrates the CVE CVSS scores for the top two CWEs. The shaded areas around each line represent the 95% confidence intervals. These intervals provide a range within which the true average CVSS score for each CWE in a given year is likely to fall, highlighting that not only is the frequency of these weaknesses increasing, but their associated severity scores are also rising. This suggests that the impact of these vulnerabilities is becoming more severe, indicating a potential escalation in their exploitability or the consequences of their exploitation. Overall, these insights emphasize the importance of addressing these recurring CWEs with effective remediation strategies to reduce their prevalence and severity. In the world of cybersecurity, two Common Weakness Enumerations (CWEs) stand out due to their prevalence and the significant risks they pose: *CWE-119 (Improper Restriction of Operations within the Bounds of a Memory Buffer) and CWE-79 (Improper Neutralization of Input During Web Page Generation, commonly known as Cross-Site Scripting or XSS).* These vulnerabilities have been documented extensively, and their associated Common Vulnerabilities and Exposures (CVEs) reveal a troubling narrative about the state of software security.

#### 4.1.5.6.1 The Tale of CWE-119

CWE-119 has a storied history, with numerous vulnerabilities dating back to the late 1990s. For example, **CVE-1999-0002** describes a buffer overflow in the NFS mount daemon, allowing remote attackers to gain root access to systems, particularly in Linux environments. This vulnerability was rated with a CVSS 2.0 Base **Score of 10**, indicating a critical severity level. The implications of such vulnerabilities are profound; they can lead to complete system compromise, allowing attackers to execute arbitrary code with elevated privileges.

As time progressed, **CWE-119** continued to manifest in various applications. By 2009, CVE-2009-0658 highlighted a buffer overflow in Adobe Reader, allowing attackers to execute arbitrary code via a specially crafted PDF document. This vulnerability also carried a CVSS score of 9.3, underscoring the ongoing threat posed by improper memory handling. The persistence of **CWE-119** in high-profile applications illustrates a systemic issue in software development practices, where memory management is often overlooked or inadequately addressed.

#### 4.1.5.6.2 The Cross-Site Scripting Saga

In contrast, **CWE-79** represents a different but equally dangerous aspect of web security. This vulnerability allows attackers to inject malicious scripts into web pages viewed by other users, leading to data theft, session hijacking, and other malicious activities. The first notable entry, CVE-2002-0270, involved an XSS vulnerability in the Opera browser, which had a CVSS score of 4.3. While this score is lower than some buffer overflow vulnerabilities, the real-world impact of XSS can be devastating, especially when exploited in high-traffic web applications.

The prevalence of CWE-79 continued into the following years, with multiple vulnerabilities reported across various platforms. For instance, CVE-2008-3457, affecting phpMyAdmin, allowed user-assisted remote attackers to inject arbitrary web scripts, demonstrating how XSS vulnerabilities can be exploited in scenarios where user interaction is required. This vulnerability had a CVSS score of 2.6, showing that while it may not be critical, the potential for exploitation remains significant, particularly in the context of poorly secured web applications.

**The Interplay of Vulnerabilities and CVSS Scores** The data reveals a troubling trend: both CWE-119 and CWE-79 have been consistently associated with high CVSS scores, indicating that they pose serious risks to software security. The high counts of occurrences at CWE-119 with 11,906 instances and CWE-79 with 26,728 instances underscore their prevalence in the software landscape. This suggests that despite advancements in security practices, these vulnerabilities continue to be a significant concern.

Moreover, the historical context provided by the CVEs associated with these CWEs highlights a critical narrative: many of these vulnerabilities have been known for years, yet they persist in modern applications. This raises questions about the effectiveness of current security practices and the need for ongoing education and awareness among developers.

#### 4.1.5.7 Relationship Between Exploited CVEs and corresponding CWEs

We examined the relationship between the top 10 most common CWEs and their corresponding CVEs to determine if these CVEs had been exploited and whether the exploit was verified or not as seen in Table (4.7). This analysis provides insight into which weaknesses are more likely to be targeted by attackers.

| N# | CWE ID | Name | Affected CVEs | Number of Exploited CVEs by status | |
|---|---|---|---|---|---|
| | | | | Verified exploited CVEs | Not verified exploited CVEs |
| 1 | CWE-79 | Improper Neutralization of Input During Web Page Generation (Cross-site Scripting) | 26728 | 1896 | 654 |
| 2 | CWE-119 | Improper Restriction of Operations within the Bounds of a Memory Buffer | 11906 | 2033 | 268 |
| 3 | CWE-89 | Improper Neutralization of Special Elements used in an SQL Command (SQL Injection) | 11356 | 2839 | 585 |
| 4 | CWE-20 | Improper Input Validation | 10446 | 733 | 187 |
| 5 | CWE-787 | Out-of-bounds Write | 10183 | 168 | 70 |
| 6 | CWE-200 | Exposure of Sensitive Information to an Unauthorized Actor | 7852 | 276 | 130 |
| 7 | CWE-22 | Improper Limitation of a Pathname to a Restricted Directory (Path Traversal) | 5902 | 1044 | 290 |
| 8 | CWE-125 | Out-of-bounds Read | 5866 | 37 | 17 |
| 9 | CWE-352 | Cross-Site Request Forgery (CSRF) | 5630 | 148 | 256 |
| 10 | CWE-416 | Use After Free | 4141 | 109 | 30 |

TABLE 4.7: Top 10 weaknesses and their exploitation status.

#### 4.1.5.8 CVE distribution by number of associated weaknesses (CWE)

The distribution of vulnerabilities by the number of associated CWEs reveals a strong pattern in our dataset of 162,095 CVEs. As shown in Figure 4.9, the vast majority (92.90%) of vulnerabilities have only one CWE associated with them, while 6.82% have two associated CWEs, and only 0.25% have three CWEs. Vulnerabilities with four or more CWEs are extremely rare, collectively accounting for less than 0.04% of the dataset, with the maximum being 7 CWEs found in a single CVE.

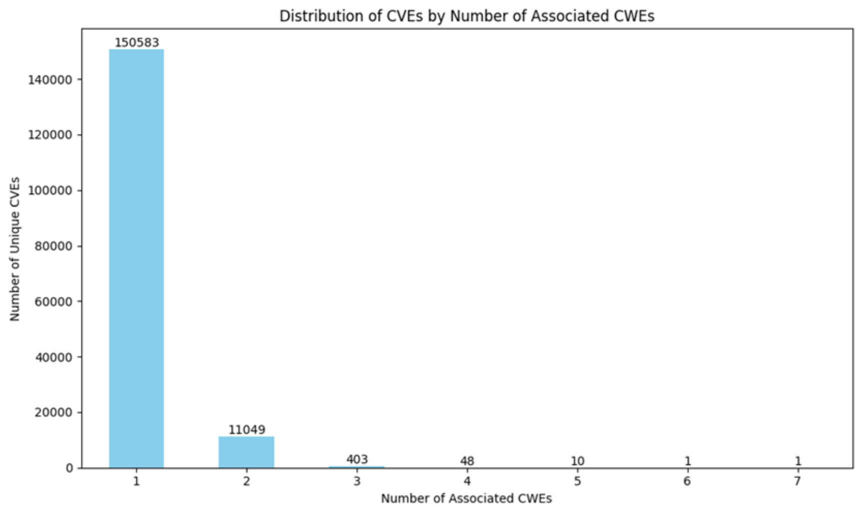


FIGURE 4.9: Distribution of CVEs by number of CWEs.

##### 4.1.5.8.1 Analysis of CWE pair per CVE patterns

Among the 11,049 CVEs with exactly two CWEs, we identified 2,024 unique CWE pairs, indicating substantial diversity in how weaknesses co-occur. Figure 4.10 shows the ten most frequently occurring CWE pairs.

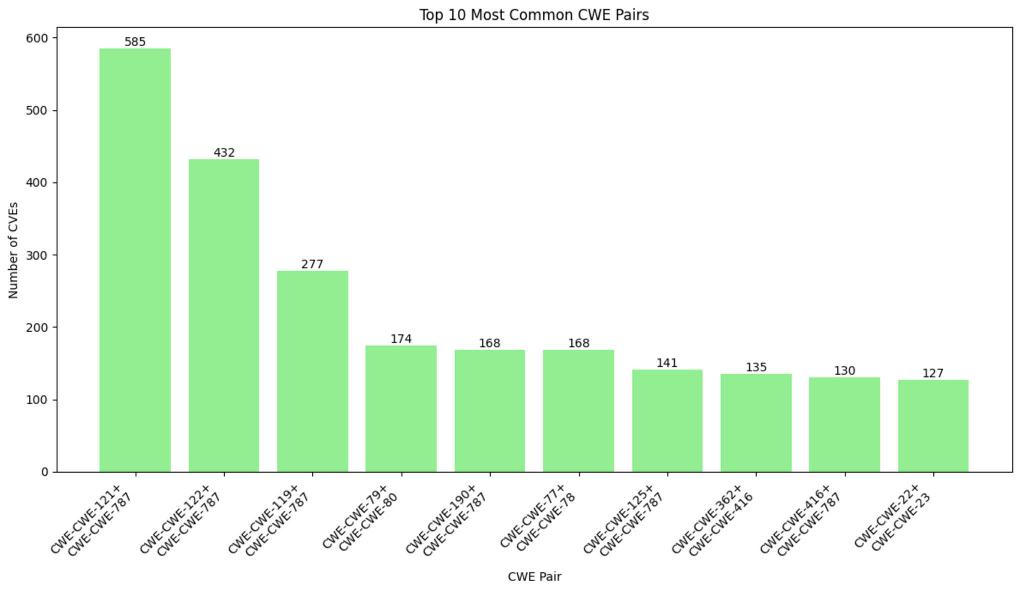


FIGURE 4.10: Top 10 most common CWE pairs found in vulnerabilities.

The most common pairs reveal important relationships between specific vulnerability types:

- **CWE-121 + CWE-787** (5.29% of all pairs): Stack-based Buffer Overflow with Out-of-bounds Write suggests memory corruption vulnerabilities that allow attackers to write beyond buffer boundaries, particularly in stack memory.
- **CWE-122 + CWE-787** (3.91%): Heap-based Buffer Overflow with Out-of-bounds Write indicates similar memory corruption issues, but affecting heap memory instead of stack memory.
- **CWE-119 + CWE-787** (2.51%): Improper Restriction of Operations within the Bounds of a Memory Buffer with Out-of-bounds Write represents a broader category of memory safety issues leading to buffer overflows.
- **CWE-79 + CWE-80** (1.57%): Cross-site Scripting (XSS) with Improper Neutralization of Script-Related HTML Tags in a Web Page highlights web vulnerabilities related to insufficient sanitization of user inputs.
- **CWE-190 + CWE-787** (1.52%): Integer Overflow or Wraparound with Out-of-bounds Write shows how arithmetic errors can lead to memory corruption.
- **CWE-77 + CWE-78** (1.52%): Improper Neutralization of Special Elements used in a Command with Improper Neutralization of Special Elements used in an OS Command reflects command injection vulnerabilities.

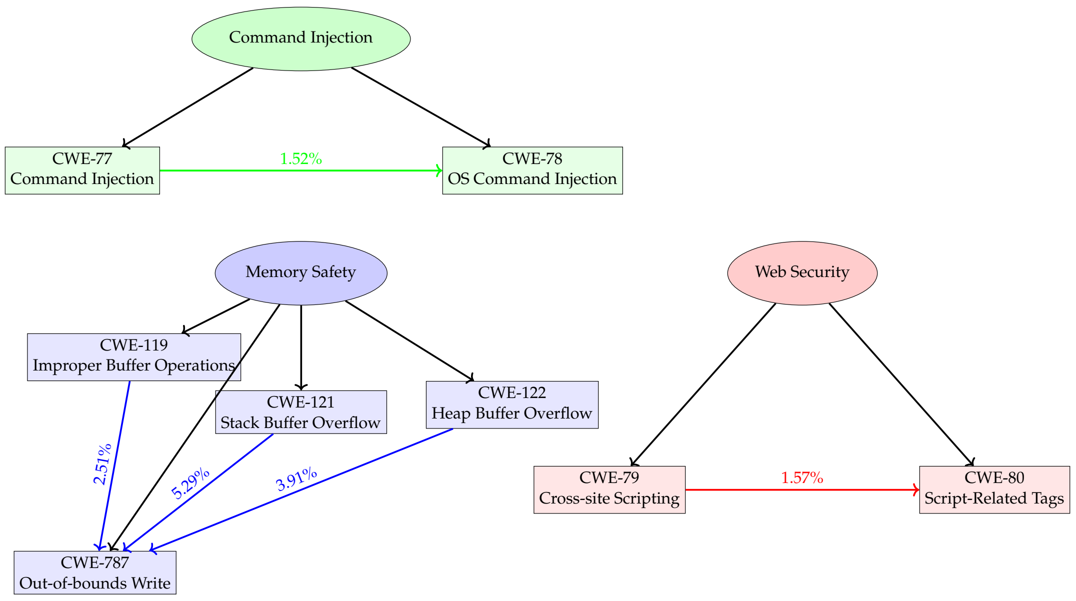


FIGURE 4.11: Relationship diagram of common CWE pairs showing the top vulnerability clusters.

Notably, CWE-787 (Out-of-bounds Write) appears in 5 of the top 10 pairs, highlighting its central role in many vulnerability combinations. This suggests that memory corruption, particularly buffer overflows that allow writing outside intended boundaries, remains a persistent and complex security challenge. Figure 4.11 provides a conceptual visualization of the most significant CWE pair relationships, organizing them into three distinct vulnerability clusters: memory safety, web security, and command injection. The diagram uses directed edges with percentage annotations to quantify the co-occurrence frequency of these vulnerability pairs, highlighting particularly strong relationships such as CWE-121 (Stack Buffer Overflow) and CWE-787 (Out-of-bounds Write) at 5.29%. This representation offers academic value by transforming tabular statistical data into a topological structure that reveals the hierarchical and relational nature of vulnerability patterns. Such visualization facilitates cognitive understanding of complex security relationships, enables identification of central vulnerability types (like CWE-787) that serve as connection points across multiple weaknesses, and provides an intuitive framework for security researchers to conceptualize vulnerability clusters as interconnected systems rather than isolated issues. This approach supports both theoretical modeling of vulnerability relationships and practical application in security education and risk assessment.

#### 4.1.5.8.2 Analysis of CWE triplet per CVE patterns

For the 403 CVEs with exactly three CWEs, we identified 314 unique triplet combinations, demonstrating even greater diversity in more complex vulnerability scenarios. Figure 4.12 shows the seven most common CWE triplets.

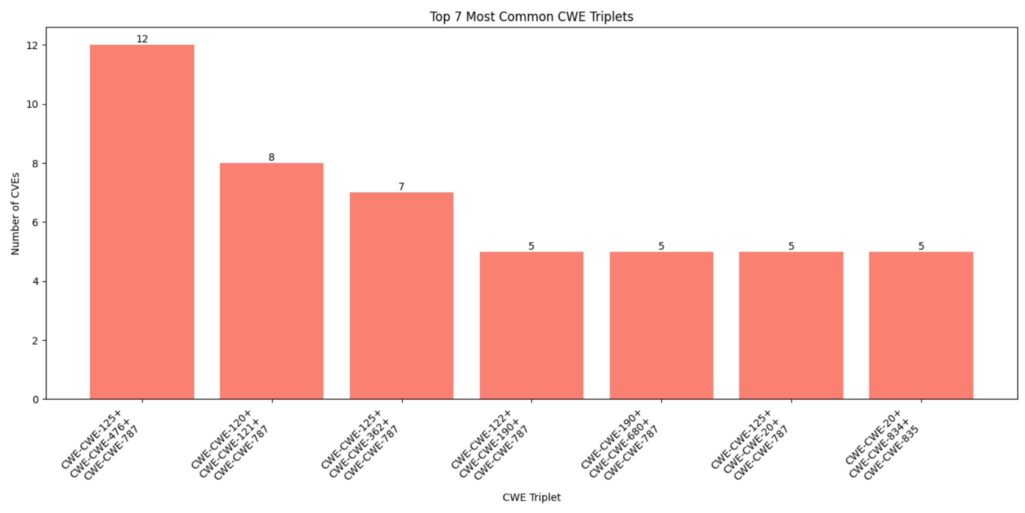


FIGURE 4.12: Top 7 most common CWE triplets found in vulnerabilities.

The most frequent triplet combinations reveal complex vulnerability relationships:

- **CWE-125 + CWE-476 + CWE-787** (2.98% of all triplets): Out-of-bounds Read with NULL Pointer Dereference and Out-of-bounds Write represents a complex scenario involving multiple memory safety issues.
- **CWE-120 + CWE-121 + CWE-787** (1.99%): Buffer Copy without Checking Size of Input with Stack-based Buffer Overflow and Out-of-bounds Write shows how insufficient input validation can lead to multiple memory corruption issues.
- **CWE-125 + CWE-362 + CWE-787** (1.74%): Out-of-bounds Read with Concurrent Execution using Shared Resource with Improper Synchronization and Out-of-bounds Write demonstrates how concurrency issues can compound memory safety vulnerabilities.

Similar to the pair analysis, CWE-787 (Out-of-bounds Write) appears in 6 of the top 7 triplets, reinforcing its significance in complex vulnerability scenarios. The triplet data also reveals that memory safety issues (CWE-125, CWE-120, CWE-121, CWE-787) frequently combine with other vulnerability types like NULL pointer issues (CWE-476) and concurrency problems (CWE-362).

#### 4.1.5.8.3 CWE (pairs, triples) pattern implications for security practices

The prevalence of specific CWE pairs and triplets provides valuable insights for security practitioners:

1. **Focus on memory safety**: The dominance of memory-related vulnerabilities (especially CWE-787) in both pairs and triplets suggests that memory safety should remain a top priority, particularly in languages like C and C++ that allow direct memory manipulation.
2. **Consider vulnerability clusters**: Rather than addressing individual CWEs in isolation, security strategies should target related vulnerability clusters, as demonstrated by the recurring pairs and triplets.

3. **Comprehensive testing approach**: Testing should account for potential vulnerability combinations, especially those involving memory corruption, web security, and command injection.

4. **Code review emphasis**: Code reviews should specifically look for patterns of co-occurring vulnerabilities as identified in the pairs and triplets analysis.

This analysis demonstrates that certain vulnerability types tend to co-occur in predictable patterns. By understanding these relationships, developers and security teams can implement more effective prevention and detection strategies targeting multiple related weaknesses simultaneously.

TABLE 4.8: CWE IDs mentioned in pairs, triplets analysis and Their Descriptions

| **CWE ID** | **Description** |
|---|---|
| CWE-20 | Improper Input Validation |
| CWE-22 | Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') |
| CWE-23 | Relative Path Traversal |
| CWE-77 | Improper Neutralization of Special Elements used in a Command ('Command Injection') |
| CWE-78 | Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') |
| CWE-79 | Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') |
| CWE-80 | Improper Neutralization of Script-Related HTML Tags in a Web Page |
| CWE-119 | Improper Restriction of Operations within the Bounds of a Memory Buffer |
| CWE-120 | Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') |
| CWE-121 | Stack-based Buffer Overflow |
| CWE-122 | Heap-based Buffer Overflow |
| CWE-125 | Out-of-bounds Read |
| CWE-190 | Integer Overflow or Wraparound |
| CWE-362 | Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') |
| CWE-416 | Use After Free |
| CWE-476 | NULL Pointer Dereference |
| CWE-680 | Integer Overflow to Buffer Overflow |
| CWE-787 | Out-of-bounds Write |
| CWE-834 | Excessive Iteration |
| CWE-835 | Loop with Unreachable Exit Condition ('Infinite Loop') |

#### 4.1.5.9 CWE Status and Weakness Abstraction

The status of a CWE indicates the maturity and completeness of its definition. The possible statuses include:

- **Stable**: Finalized and reviewed definitions, widely used for vulnerability identification and analysis.

- **Draft**: Initial definitions still under development, used for informational purposes but not yet finalized.

- **Incomplete**: Basic definitions missing crucial details, placeholders for future development.

- **Deprecated**: Obsolete weaknesses due to changes in technology or security practices, not actively maintained. The below are examples of deprecated CWEs:

  – **CWE-477: Use of Obsolete Function:** This CWE described the use of deprecated or obsolete functions in code. It was deprecated because the concept of "obsolete" is subjective and can vary depending on the context.

    * **Reason for deprecation:** Determining whether a function is truly obsolete can be difficult, and the CWE didn't provide clear criteria for making this distinction.
  - **CWE-609: Double-Free:** This CWE referred to a vulnerability where a program attempts to free the same memory location twice. This can lead to crashes or unpredictable behavior.
    * **Reason for deprecation:** Modern memory management systems often have safeguards against double-frees, making this CWE less relevant.

**CWE weaknesses can also be categorized based on their abstraction level as per MITRE** [107], which refers to the specificity of the description:

- **Pillar Weakness**: The most general category, describing vulnerabilities with one or two key characteristics. An example for it is **CWE-20: Improper Input Validation**.
- **Class Weakness**: More detailed, defining vulnerabilities using two or three dimensions, independent of specific technologies. An example for it is **CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer**.
- **Base Weakness**: Introduces more specific details associated with groups of technologies or resources. An example for it **CWE-120: Classic Buffer Overflow**.
- **Variant Weakness**: Highly specific, linked to particular languages, technologies, or resources. An example for it is **CWE-79: Improper Neutralization of Input During Web Page** Generation ('Cross-site Scripting').
- **Compound Weakness**: Aggregations of several weaknesses, known as either a Chain or Composite. An example for it is **CWE-400: Uncontrolled Resource Consumption**: This weakness describes a scenario where a resource is consumed without any limits, potentially leading to denial of service. It can be a chain of multiple weaknesses, such as CWE-770 (Allocation of Resources Without Limits or Throttling) followed by CWE-401 (Missing Release of Memory after Effective Lifetime).

The below graph 4.13 describes how the distribution of Common Weakness Enumerations (CWEs) in vulnerabilities (CVEs) has changed over time. It considers both the CWE abstraction level (class, base, variant, etc.) and the CWE status (draft, stable, incomplete). Our data suggests that most CWEs are in a *Stable* status, which suggests CVEs mapped to stable CWEs, indicating wider acceptance and maturity. The most abstraction level increased over time is *Base* level, which means more details regarding the weakness are improved dramatically. The CWE data appeared in CVEs since 2004 and has improved yearly to a more specific and accurate state.

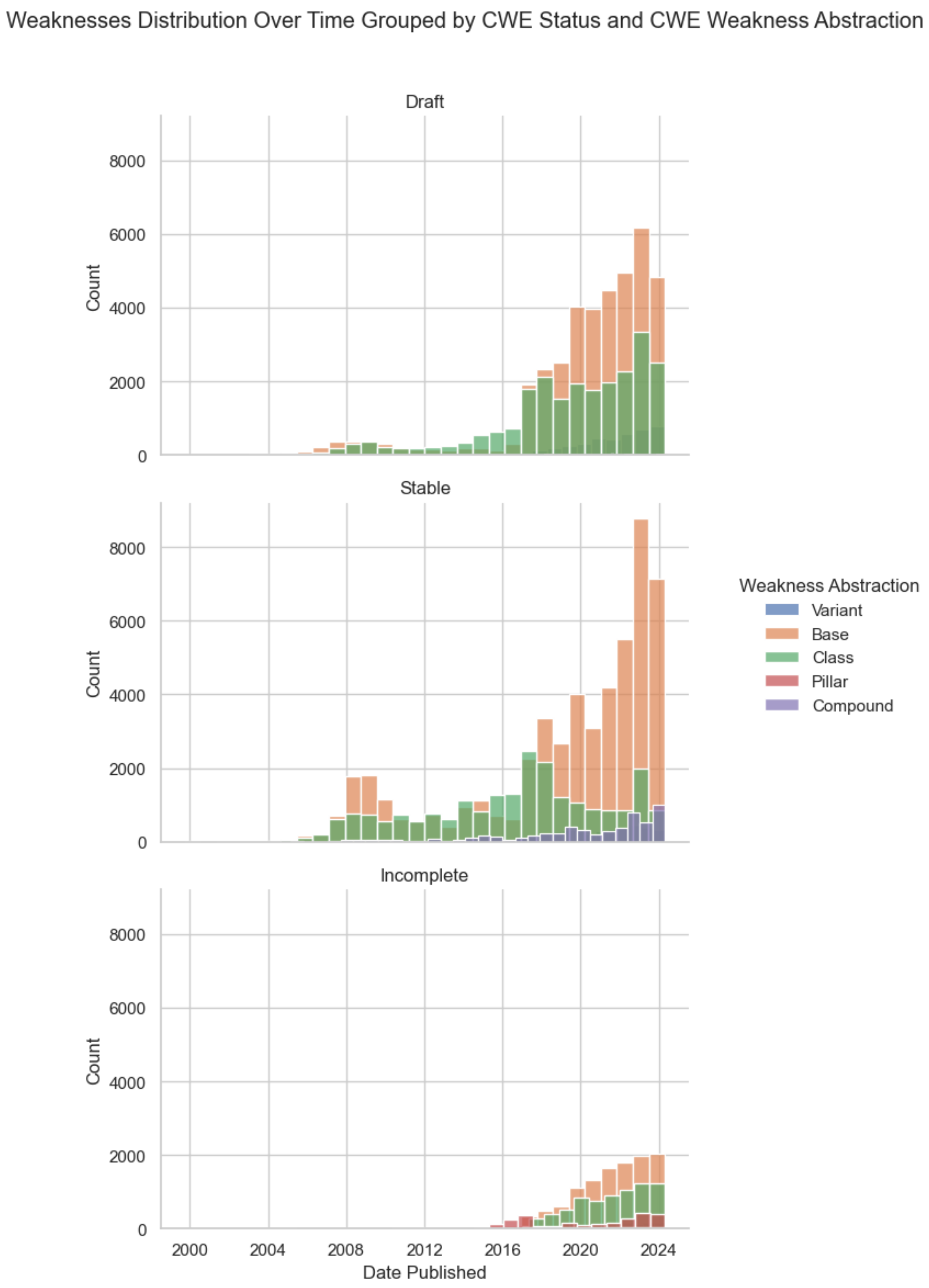


FIGURE 4.13: Distribution of CWEs by status and abstraction level.

#### 4.1.5.10 Detailed Analysis of Selected CVEs with Multiple CWEs

We conducted a detailed investigation of CVEs with five or more associated CWEs to understand the complexity and interrelated nature of these vulnerabilities. Out of 12 CVEs with 5 or more CWEs, we analyzed a subset of these CVEs as shown below:

- CVE-2019-10084: Associated CWEs include CWE-311, CWE-330, CWE-384, CWE-532, CWE-732. This vulnerability in Apache Impala involves issues such as improper handling of session IDs and insufficient random values.

- CVE-2024-1848: Associated CWEs include CWE-125, CWE-416, CWE-787, CWE-843, CWE-908. Vulnerabilities in SOLIDWORKS Desktop lead to arbitrary code execution.
- CVE-2023-46724: Associated CWEs include CWE-125, CWE-1285, CWE-129, CWE-295, CWE-786, CWE-823. Denial of Service attacks in Squid due to improper validation of array indices.
- CVE-2018-1000224: Associated CWEs include CWE-131, CWE-190, CWE-681, CWE-908, CWE-909. Issues in Godot Engine's (De)Serialization functions.
- CVE-2020-15098: Associated CWEs include CWE-20, CWE-200, CWE-325, CWE-327, CWE-502. Vulnerabilities in TYPO3 CMS leading to privilege escalation and insecure deserialization.
- CVE-2021-26726: Associated CWEs include CWE-209, CWE-272, CWE-305, CWE-330, CWE-78. Remote code execution in Valmet DNA service due to vulnerabilities on TCP port 1517.

#### 4.1.5.10.1 The common characteristics of the CVEs with multiple CWEs

- The earliest date for the above CVE is July 2018. The latest has been reserved for February 2024 and updated on April 2024.
- The average CVSS score is 8.3, which means that most of the above CVEs are critical and this is normally due to the high number of associated CWEs.
- 66% of the above CVEs are affecting only one CPE (product), 17% has no information regarding the affected products, and 17% is affecting more than one CPE.

If we take the last CVE from the above list CVE-2021-26726. This vulnerability is a combination of issues related to authentication and privilege management. Information exposure (CWE-209), improper privilege assignment (CWE-272), and authentication bypass (CWE-305) allow attackers to gain SYSTEM privileges. Insufficiently random values (CWE-330) and OS command injection (CWE-78) further enable remote code execution. Vulnerabilities often manifest through multiple weaknesses, and understanding the relationship between these weaknesses can provide deeper insights into potential exploitation pathways. CVEs associated with multiple CWEs indicate complex vulnerabilities that may involve various layers of an application, often making them harder to detect and mitigate.

These CVEs involve complex vulnerabilities that encompass multiple CWEs due to the interrelated nature of the flaws. They often stem from improper validation, insecure cryptographic practices, and incorrect handling of memory or data, which can lead to severe consequences such as remote code execution, privilege escalation, and denial of service. Addressing these vulnerabilities requires comprehensive fixes that cover all related CWEs to ensure the security and stability of the affected systems.

Key Insights from CVE Analysis

1. **Vulnerability Disclosure Trends:**
   - Significant peaks in CVE disclosures observed during 2017-2021 and 2022-2024
   - 78% increase during the pandemic period (2020-2021), exceeding the 10% annual growth predicted by industry reports
   - Projected 547 new CVEs weekly by 2025, with some weeks reaching 1,250 new disclosures
2. **Severity Distributions:**
   - Medium-severity vulnerabilities predominate (50.9%), followed by high (28.0%), low (14.1%), and critical (7.1%)
   - No significant increase in average severity over time despite the introduction of CVSSv3
   - Apparent severity increases more attributable to changes in scoring methodology than to fundamental changes in vulnerability characteristics
3. **Weakness Persistence:**
   - CWE-79 (Cross-site Scripting) remains the most prevalent weakness (15.35% of all CWEs) despite decades of awareness
   - Memory safety issues (CWE-787, CWE-416, CWE-119, CWE-125) collectively account for a substantial portion of vulnerabilities
   - Significant peaks in 2019-2021 for nearly all top CWEs, followed by collective decline in 2022
4. **Complex Vulnerability Patterns:**
   - 93% of CVEs have only one associated CWE, while 7% exhibit more complex patterns with multiple weaknesses
   - Memory-related vulnerabilities dominate multi-CWE relationships, with CWE-787 (Out-of-bounds Write) appearing in most common pairs and triplets
   - CVEs with five or more CWEs average a CVSS score of 8.3, indicating that complexity often correlates with severity
5. **Exploitation Implications:**
   - SQL Injection (CWE-89) shows the highest exploitation rate (25.0
   - Web vulnerabilities like Cross-site Scripting remain prevalent but show lower exploitation rates (9.5
   - Memory safety issues in the top CWEs show varying exploitation rates, from 1.7

### 4.1.6 Trends and Insights from CVE Analysis

The analysis of CVE data reveals several important trends:

- **Increasing Number of CVEs**: The number of CVEs reported annually has increased over the years, reflecting heightened awareness and reporting as outlined in Figure 4.1.
- **Severity Distribution**: The majority of CVEs fall under the medium severity category as seen in 4.2, highlighting a significant portion of vulnerabilities that pose a moderate risk.
- **Common Weaknesses**: The prevalence of certain CWEs, such as Cross-site Scripting (CWE-79) and SQL Injection (CWE-89) outlined in Figure 4.6, indicates persistent issues in software security practices.

The importance of understanding CVEs:

> The systematic identification and classification of vulnerabilities through CVEs are fundamental to improving security measures and mitigating risks.
>
> Analyzing the patterns and trends in CVE data can provide valuable insights into the evolving landscape of cybersecurity threats.

## 4.2 Exploits

An exploit is a piece of software, a chunk of data, or a sequence of commands that takes advantage of a bug or vulnerability to cause unintended or unanticipated behavior to occur on computer software, hardware, or something electronic (usually computerized). This often includes such things as gaining control of a computer system, allowing privilege escalation, or a denial-of-service attack.

In the context of cybersecurity, exploits are a crucial aspect to consider as they represent the practical application of vulnerabilities that have been discovered. An exploit typically takes advantage of a vulnerability to perform unauthorized actions on a system. These actions could range from stealing data, and executing malicious code, to disabling systems entirely. The availability of an exploit often accelerates the necessity for patching the associated vulnerability, as it can significantly increase the risk of attacks.

### 4.2.1 Exploit Database (ExploitDB) Data

The Exploit Database (ExploitDB) [112] is a CVE-compliant archive of public exploits and corresponding vulnerable software, developed for use by penetration testers and vulnerability researchers. It provides a comprehensive collection of exploits, ranging from remote code execution to local privilege escalation.

In our dataset, the *exploit* table contains detailed information on exploits obtained from the ExploitDB GitHub repository [112]. This table includes the following attributes:

- **ID**: Unique identifier for the exploit record.
- **Exploit ID**: Unique identifier for the specific exploit.

- **File**: Path to the file containing exploit details.
- **Description**: Brief description of the exploit.
- **Date Published**: Date when the exploit was published.
- **Author**: Author of the exploit.
- **Type**: Type of exploit (e.g., webapps, local).
- **Platform**: Target platform of the exploit.
- **Date Added**: Date when the exploit was added to the database.
- **Date Updated**: Date when the exploit was last updated.
- **Verified**: Indicates if the exploit is verified.
- **Codes**: References such as CVE and OSVDB IDs.
- *CVE ID*: CVE ID is retrieved from the above *Codes* column.
- **Source URL**: URL to the source of the exploit information.

| **Table** | **Number of Records** |
|---|---|
| exploit (Exploit Data) | 46,494 |

TABLE 4.9: Overview of the ExploitDB dataset.

#### 4.2.1.1 Insights from the ExploitDB Data

The data from ExploitDB provides significant insights into the nature and trends of exploits in the wild. For instance, the distribution of exploits over time in Graph 4.14 shows that certain years have higher incidences of exploit publication, which could correlate with increased discovery of vulnerabilities or advancements in exploit development techniques. In addition, it shows the distribution of exploit verification across different years. This will help us identify if exploit verification is becoming more or less prevalent over time.

### 4.2.2 Relationship Between Exploits and CVEs

The relationship between exploits and CVEs is critical, as it helps in understanding which vulnerabilities are actively targeted by attackers. By mapping exploits to their corresponding CVEs, we can assess the risk associated with specific vulnerabilities. For instance, a CVE with multiple associated exploits indicates a higher risk due to the availability of numerous attack vectors.

Overall, the data from ExploitDB not only provides a historical archive of exploits but also serves as a practical resource for understanding current and emerging threats. This information is vital for developing effective vulnerability management and mitigation strategies.

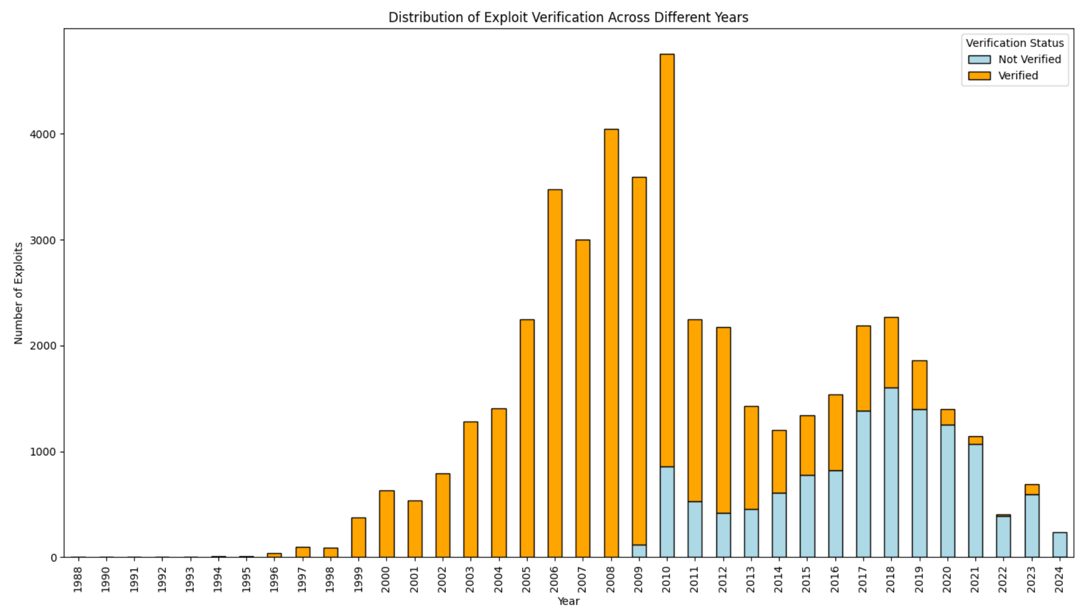


FIGURE 4.14: Annual distribution of published exploits with their status.

The above Figure 4.14 suggests a significant increase in the number of exploits discovered over the years. This trend aligns with the general growth of software development and the increasing complexity of software systems. This trend highlights the ongoing challenge of securing software in a constantly evolving technological landscape. By focusing on some specific observations from ExploitDB:

- **Early Years (1988-1999)**: The number of exploits remained relatively low.
- **Rapid Rise (1999-2006)**: A dramatic increase in exploits occurred, potentially due to the widespread adoption of the internet and the rise of readily available attack tools.
- **Fluctuations (2007-2020)**: The data shows some stabilization or even a decrease in verified exploits after 2006, possibly due to increased security awareness and improved software development practices. However, the total number of exploits (verified and unverified) continues to rise.
- **Recent Decline (2021-2024)**: There's a significant decrease in both verified and unverified exploits in recent years. This could be due to several factors, like increased focus on exploit verification, leading to a higher proportion of unverified exploits being filtered out, and improved security measures making vulnerabilities harder to exploit. A shift towards more targeted attacks with less focus on publicly available exploits. The source we are using has fewer exploits being reported, the exploits data are in other sources.
- **Verified vs Unverified Exploits**: The number of unverified exploits consistently exceeds verified exploits. This could be because verifying exploits requires time and resources, and not all discovered exploits are thoroughly checked. Attackers might withhold exploit details to maintain an advantage.

### 4.2.3 The Exploit types and statuses

Analyzing the types of exploits reveals that web application exploits are the most prevalent, reflecting the widespread use of web technologies and the common associated vulnerabilities. This is followed by local exploits, which typically require local access to the vulnerable system but can lead to significant privilege escalation if exploited successfully. ExploitDB categorizes exploits as "verified" or "non-verified". Verified exploits have been tested and confirmed to work in a controlled environment. Non-verified exploits have not been tested by the ExploitDB team but are still considered valuable for research and development purposes.

Based on the Figure 4.15 showing the counts of verified and unverified exploits categorized by type, several insights can be drawn:

- **Overall Distribution:** The graph provides a breakdown of exploits into different types, including **"dos"** (Denial of Service), **"local"**, **"remote"** and **"webapps"**. Each type represents a different category of exploit techniques.
- **Frequency of Exploits:** The graph shows that "webapps" have the highest frequency of both verified (9937) and unverified (2583) exploits. This suggests that web application vulnerabilities are commonly targeted by attackers and have a significant presence in the dataset.
- **Variability in Verification Status:** Across all exploit types, the number of verified exploits is notably higher than unverified ones. This could indicate a higher success rate in validating and confirming exploits, possibly due to more reliable sources or clearer evidence of exploitation.
- Difference in Verification Status by Type:
  - **Denial of Service (dos):** This category has a higher number of verified exploits (3302) compared to unverified ones (462). This might suggest that Denial of Service attacks are more easily identifiable and verifiable, leading to a higher verification rate.
  - **Local Exploits:** The number of verified local exploits (1621) is higher than unverified ones (405), indicating a 1/4 higher success rate in confirming local exploitation techniques. This could be due to the complexity of detecting local vulnerabilities or the lack of reliable indicators for verification.
  - **Remote Exploits:** Similar to Denial of Service attacks, remote exploits have a higher number of verified cases (3413) compared to unverified ones (408), indicating a relatively higher success rate in verifying remote exploitation techniques.
  - **Web Application (webapps) Exploits:** Web application exploits exhibit the highest disparity between verified (9937) and unverified (2583) cases. This suggests that web application vulnerabilities are both prevalent and more easily identifiable and confirmable compared to other exploit types. The significant prevalence of web application exploits (9,937 verified cases) in our dataset demonstrates Meunier's [102] argument about the multidimensional nature of vulnerability classification. As Meunier observed, vulnerabilities exist at multiple abstraction levels simultaneously, and our findings on web application exploit dominance highlight how these vulnerabilities persist despite being well-documented, precisely because they

span implementation, design, and deployment abstraction layers, making them particularly resistant to comprehensive remediation.

- **Implications for Security:** Understanding the distribution and verification status of exploits by type can provide valuable insights for cybersecurity professionals and organizations. It highlights the importance of prioritizing security measures, such as patching and vulnerability management, particularly for web applications and remote exploitation techniques, which appear to be more prevalent and verifiable. Additionally, it underscores the need for further research and detection capabilities in identifying and validating local exploits, where verification rates appear to be near to unverified by 1 to 4.

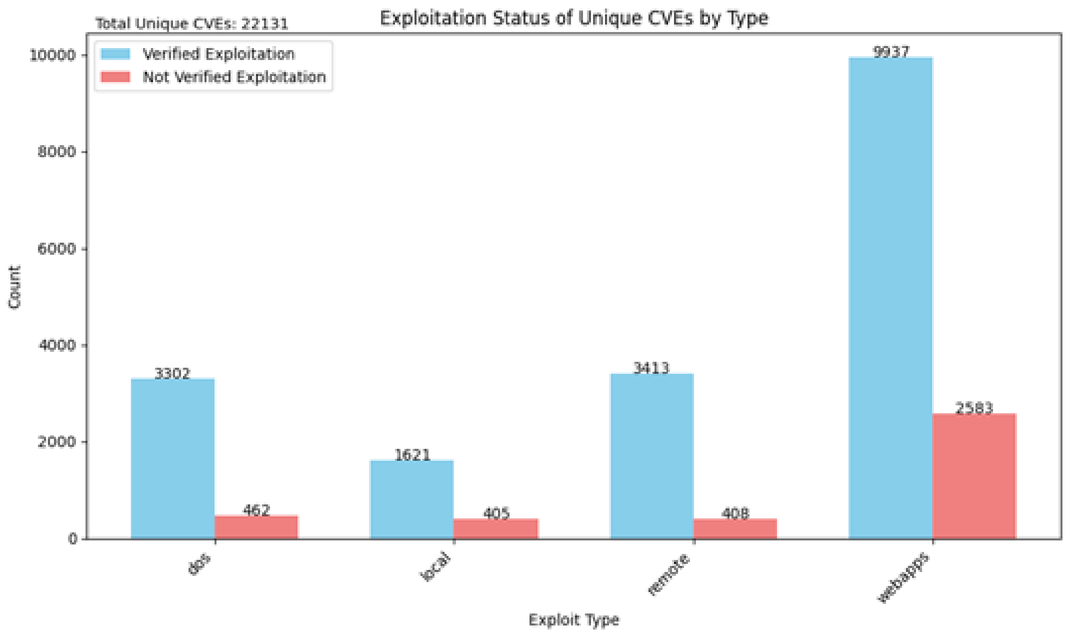


FIGURE 4.15: Distribution of Exploit types and their status.

Our analysis shows that a significant portion of the exploits in the database are verified, which underscores the reliability of the data provided by ExploitDB. This verification process helps in distinguishing functional exploits from theoretical ones, aiding security professionals in prioritizing their response efforts.

### 4.2.4 Distribution of Exploits per CVE

Analyzing the distribution of exploits per CVE revealed significant insights. Most CVEs have between 1 to 2 exploits on average, with some outliers having numerous exploits. In our data, there are 12 CVEs with 13 or more exploits. For instance, *CVE-2012-1199* has 38 associated exploits which we will discuss later.

Understanding the distribution of exploits per Common Vulnerabilities and Exposures (CVE) provides valuable insights into the security landscape. The analysis of the dataset reveals the frequency and patterns of exploits associated with CVEs, which can help in prioritizing patch management and vulnerability mitigation efforts.

Figure 4.16 illustrates the number of CVEs categorized by the number of associated exploits. A significant majority of CVEs have only one associated exploit, while a smaller number have multiple exploits. This distribution highlights that although a single vulnerability may not always lead to multiple exploit attempts, those that do are likely to be more critical and heavily targeted by attackers.

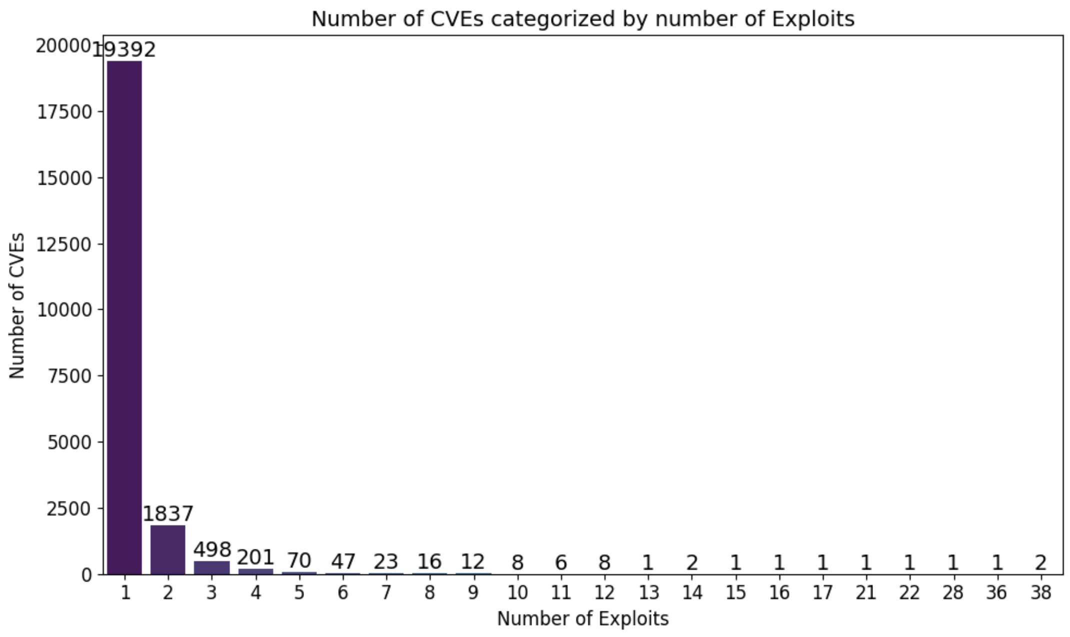


FIGURE 4.16: Number of CVEs categorized by number of Exploits

Figure 4.17 compares the number of CVEs with one exploit against those with more than one exploit. The analysis shows that 19,392 CVEs have a single exploit, whereas 2,738 CVEs have more than one exploit. This stark difference underscores the fact that while the presence of multiple exploits per CVE is relatively rare, such instances represent a significant security risk due to the increased likelihood of exploitation.

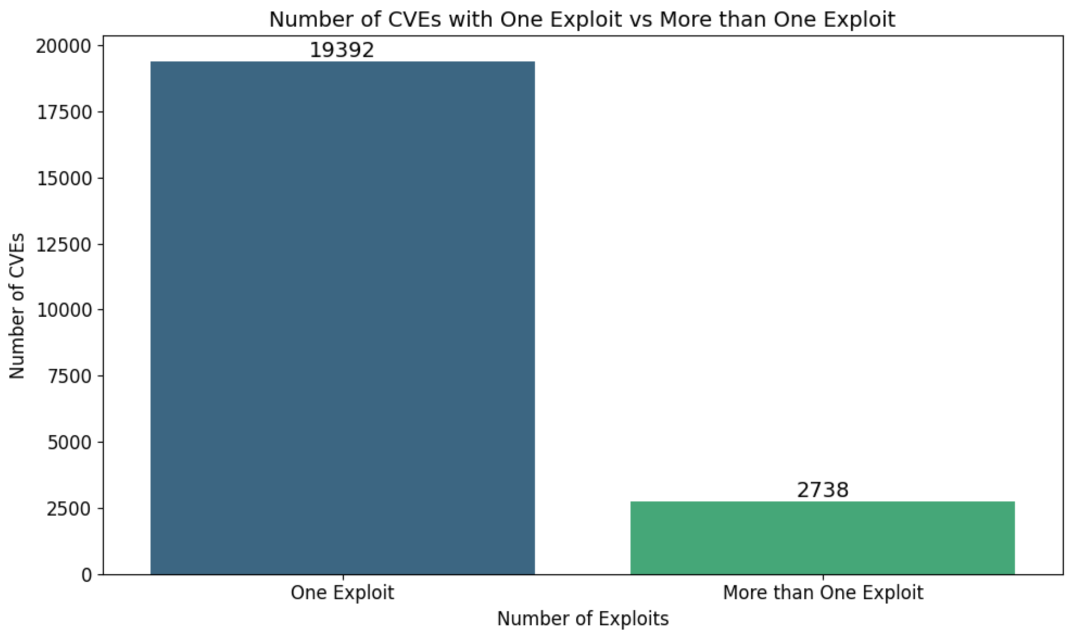


FIGURE 4.17: Number of CVEs with One Exploit vs More than One Exploit

These findings suggest that security teams should focus not only on patching vulnerabilities but also on monitoring and mitigating those CVEs that attract multiple exploits. The data indicates that these CVEs could be of higher importance due to their attractiveness to attackers, which aligns with broader trends observed in cybersecurity research.

These observations reinforce the importance of a comprehensive vulnerability management strategy that incorporates both the frequency and severity of exploits associated with CVEs.

### 4.2.5 Relationship between Exploitation and CPE

The exploitation rate of Common Platform Enumerations (CPE) provides critical insights into how vulnerabilities are being leveraged across different vendors. Exploitation rates are calculated as the ratio of exploited vulnerabilities to the total reported CVEs for a vendor. This analysis highlights key patterns in targeting and exploitation trends across major vendors.

For instance, the exploitation rates for leading vendors are:

- **Microsoft**: 9.13% (1,747 exploited CVEs out of 19,139 total CVEs).
- **Linux**: 7.65% (459 exploited CVEs out of 5,999 total CVEs).
- **Apple**: 6.37% (712 exploited CVEs out of 11,169 total CVEs).

Key insights include:

- Vendors with a larger attack surface, such as Microsoft and Linux, show higher exploitation rates due to their widespread use in enterprise environments.
- Enterprise-focused vendors tend to have higher exploitation rates compared to consumer-focused vendors, such as Google.

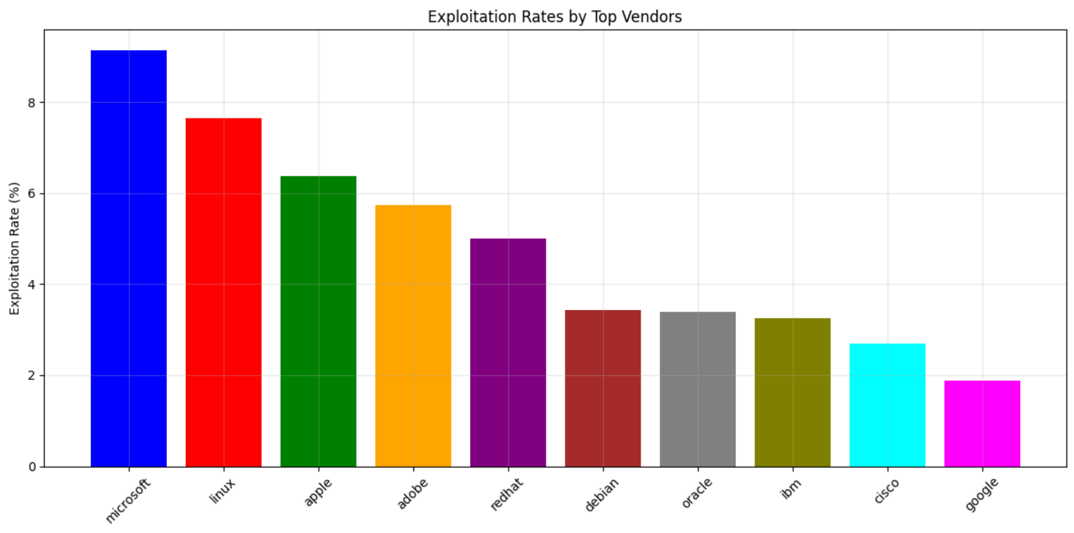


FIGURE 4.18: Exploitation rates of CPEs for major vendors. Higher rates for enterprise-focused vendors like Microsoft and Linux reflect their significant attack surface and critical system usage.

In addition, a timeline analysis was performed for the top 10 vendors based on exploited CVEs. Figure 4.19 illustrates a line chart that visualizes the annual trends of exploited CVEs for these top 10 vendors. This temporal perspective reveals periods of heightened exploitation activity and shifts in attacker focus, which may correspond to significant product changes, patch releases, or emerging vulnerabilities.

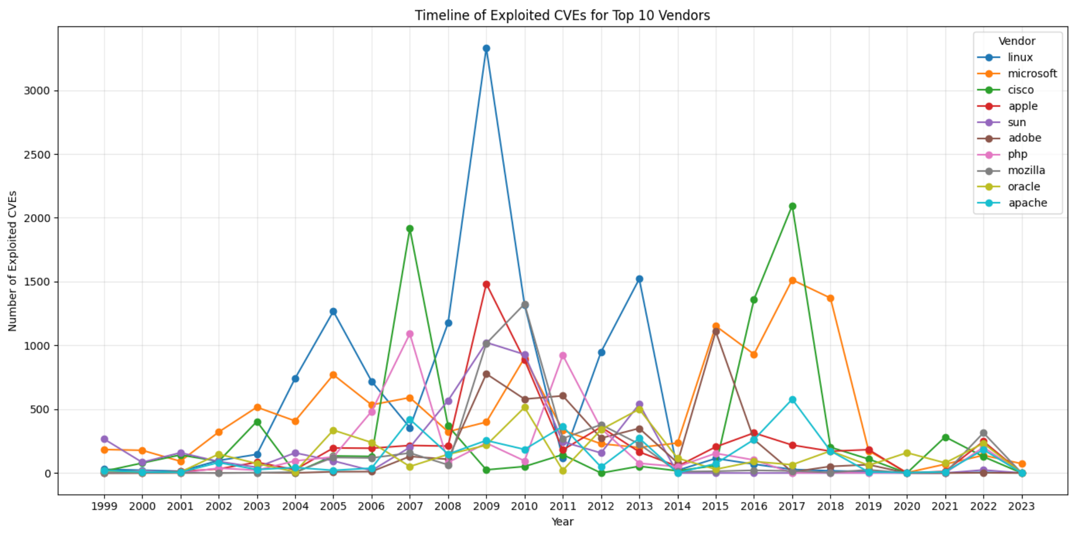


FIGURE 4.19: Timeline of exploited CVEs for the top 10 vendors. The line chart depicts annual exploited CVE counts, highlighting trends and temporal shifts in exploitation across vendors.

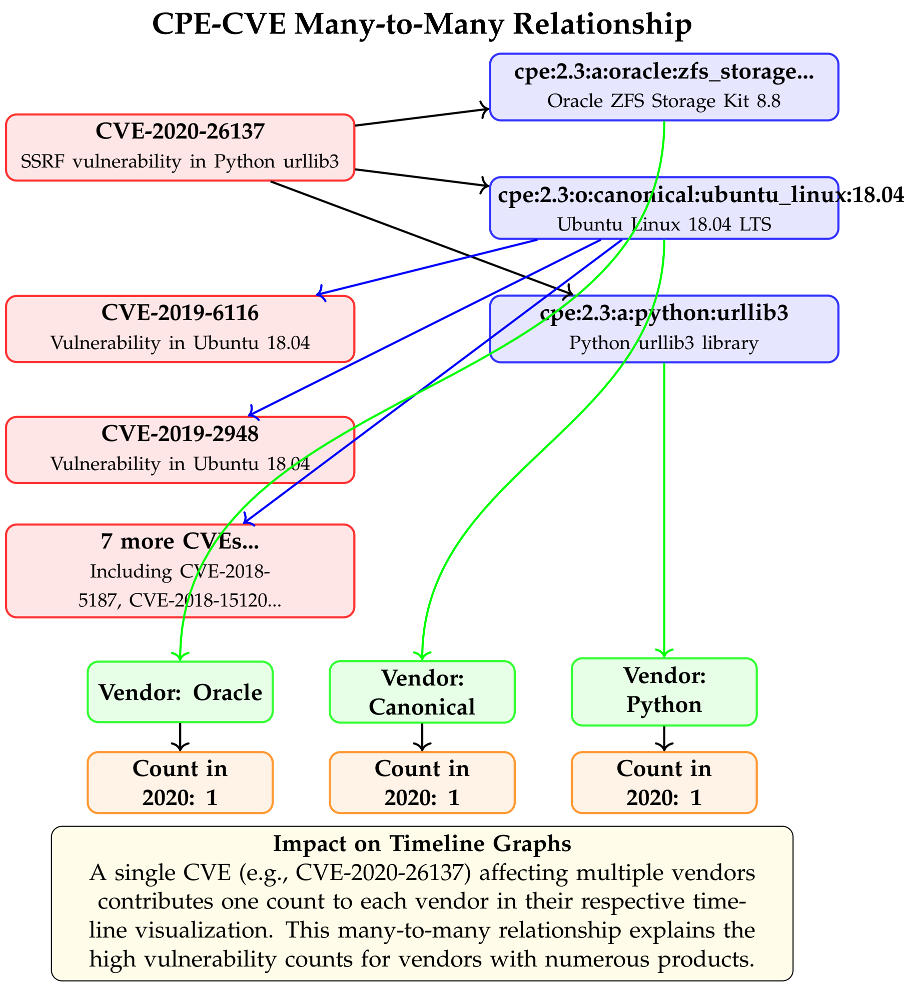

CPE-CVE Many-to-Many Relationship
cpe:2.3:a:oracle:zfs_storage...
Oracle ZFS Storage Kit 8.8
CVE-2020-26137
SSRF vulnerability in Python urllib3
cpe:2.3:o:canonical:ubuntu_linux:18.04
Ubuntu Linux 18.04 LTS
CVE-2019-6116
Vulnerability in Ubuntu 18.04
cpe:2.3:a:python:urllib3
Python urllib3 library
CVE-2019-2948
Vulnerability in Ubuntu 18.04
7 more CVEs...
Including CVE-2018-5187, CVE-2018-15120...
Vendor: Oracle
Vendor: Canonical
Vendor: Python
Count in 2020: 1
Count in 2020: 1
Count in 2020: 1
Impact on Timeline Graphs
A single CVE (e.g., CVE-2020-26137) affecting multiple vendors contributes one count to each vendor in their respective timeline visualization. This many-to-many relationship explains the high vulnerability counts for vendors with numerous products.

FIGURE 4.20: Illustration of the many-to-many relationship between CVEs and CPEs, showing how vulnerability counts are derived for vendor timeline visualization. This example demonstrates how a single CVE affects multiple vendor products, and how a single product (Ubuntu 18.04) is affected by multiple CVEs.

It is important to note that this data outlined in Figure 4.19 was extracted by analysing the Common Platform Enumeration (CPE) fields within the CVE database. The methodology outlined in Figures [4.20 and 4.21] involved parsing vendor information from CPE strings and associating them with corresponding CVEs. This approach presents several methodological considerations: a single CVE may affect multiple vendors through different CPEs, and conversely, a single CPE may appear in multiple CVEs. This many-to-many relationship between CVEs and CPEs explains the exceptionally high numbers observed for some vendors, as a single vulnerability affecting a common component might be counted multiple times across different products from the same vendor.

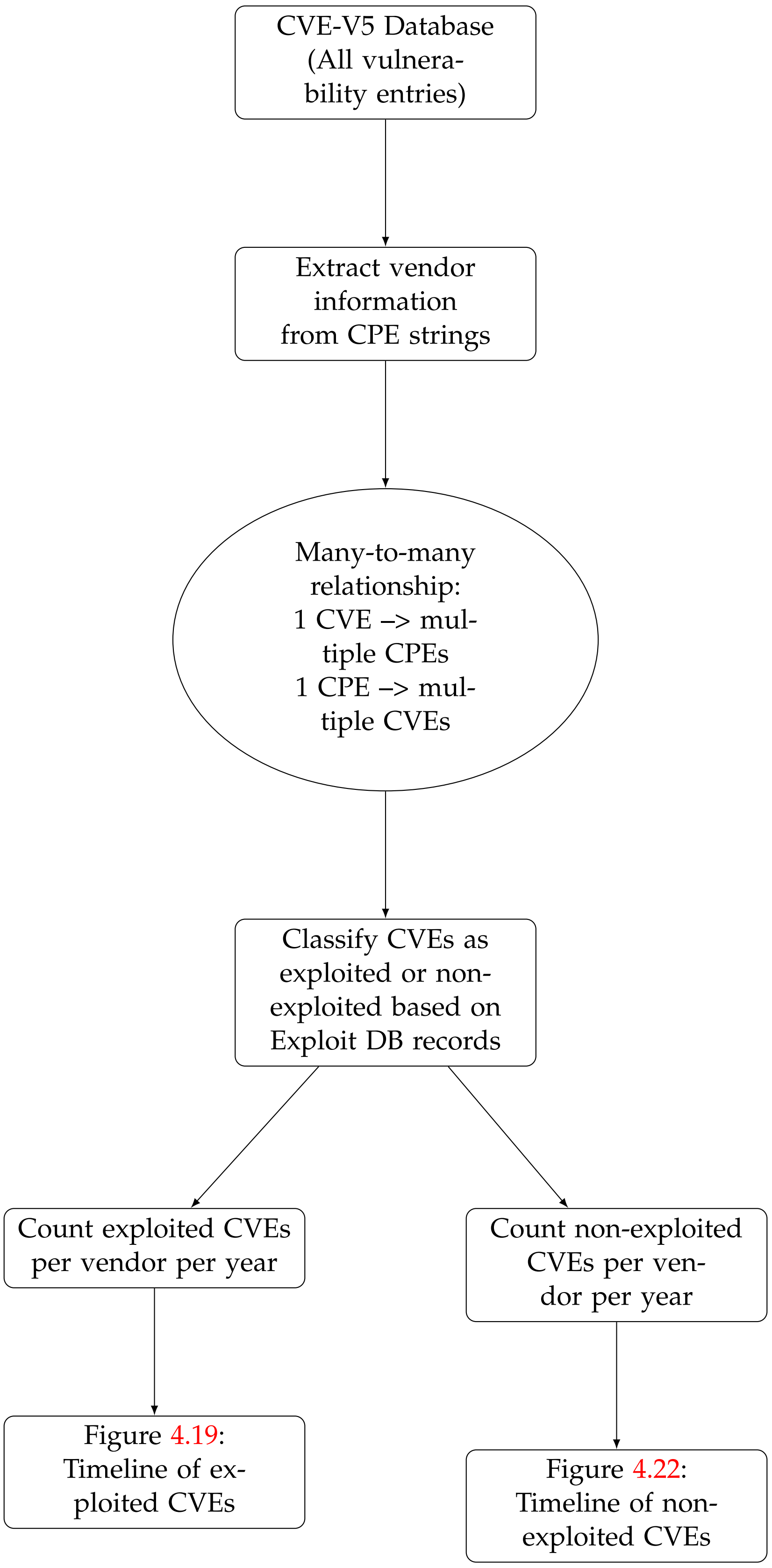

CVE-V5 Database (All vulnerability entries)
Extract vendor information from CPE strings
Many-to-many relationship: 1 CVE –> multiple CPEs 1 CPE –> multiple CVEs
Classify CVEs as exploited or non-exploited based on Exploit DB records
Count exploited CVEs per vendor per year
Count non-exploited CVEs per vendor per year
Figure 4.19: Timeline of exploited CVEs
Figure 4.22: Timeline of non-exploited CVEs

FIGURE 4.21: Data processing workflow for vendor exploitation analysis

**Insights from Timeline Exploitation Analysis** The timeline analysis of exploited CVEs for the top 10 vendors reveals the following key trends:

- **Persistent Targeting:** Microsoft consistently shows high exploitation numbers, with annual counts often exceeding 400 exploited CVEs. Linux also demonstrates notable growth, peaking at over 1,500 in certain years.
- **Variable Peaks:** Some vendors exhibit spikes; for example, Cisco and Oracle show sudden increases (e.g., Cisco exceeding 2,000 exploited CVEs in peak years) that suggest reactive attacks following major vulnerability disclosures.
- **Exploitation Rates:** Enterprise-focused vendors like Microsoft have an exploitation rate of 9.13%, while Linux and Apple have rates of 7.65% and 6.37%, respectively, highlighting their large attack surfaces.
- **Evolving Trends:** Despite fluctuations, the overall trend is upward for major vendors, indicating an evolving threat landscape with increased exploitation over time.

#### 4.2.5.1 Methodology: Parsing CPE and CVE Relationships

To accurately understand the distribution of vulnerabilities across vendors, we employed a methodical approach for extracting and analyzing the relationships between Common Vulnerabilities and Exposures (CVEs) and Common Platform Enumeration (CPE) identifiers. Figure 4.20 illustrates the many-to-many relationship that exists between these entities and explains how this affects our vulnerability count visualizations. The CPE standard (version 2.3) uses a structured string format that encodes various attributes of IT systems, software, and packages. For example, the CPE string:

```
cpe:2.3:o:canonical:ubuntu_linux:18.04::::lts:::
```

represents Ubuntu Linux 18.04 LTS operating system from Canonical. By parsing these CPE strings, we extract critical information including:

- **Part**: Whether the component is an application (`a`), operating system (`o`), or hardware (`h`)
- **Vendor**: The organization that developed the product (e.g., `canonical`, `oracle`, `python`)
- **Product**: The specific product name (e.g., `ubuntu_linux`, `zfs_storage_appliance_kit`)
- **Version**: The specific version affected (e.g., `18.04`, `8.8`)

Our analysis revealed important complexities in the CPE-CVE relationship that significantly impact vulnerability counts:

1. **One-to-Many CVE-to-CPE**: A single vulnerability (CVE) typically affects multiple products across different vendors. For example, CVE-2020-26137 (a Server-Side Request Forgery vulnerability in Python's urllib3 library) affects at least 7 different CPEs spanning multiple vendors including Oracle, Canonical, and Python itself.

2. **One-to-Many CPE-to-CVE**: Conversely, a single product (represented by a CPE) is often affected by multiple vulnerabilities. The Ubuntu Linux 18.04 LTS CPE (`cpe:2.3:o:canonical:ubuntu_linux:18.04:*:*:*:lts:*:*:*`) is associated with at least 10 different CVEs including CVE-2020-26137, CVE-2019-6116, and CVE-2019-2948.

3. **Counting Implications**: In our temporal analysis of vulnerability trends by vendor, we count each unique CVE-CPE combination. This approach generates multiple counts from a single CVE when it affects multiple CPEs from the same vendor, which explains the extraordinarily high counts observed for vendors like Qualcomm, which has extensive product lines with shared components.

This methodological approach provides a more comprehensive view of a vendor's vulnerability exposure than simply counting unique CVEs, as it captures the breadth of impact across a vendor's product portfolio. However, it also explains the dramatic disparity between our vendor vulnerability counts and the total number of unique CVEs in the database. For example, while the CVE database contains approximately 246,000 unique vulnerabilities, our analysis shows Qualcomm alone with nearly 100,000 non-exploited vulnerability instances in 2021-a reflection of this many-to-many relationship rather than an indication that Qualcomm accounts for an impossible proportion of all vulnerabilities. This enriched analysis not only confirms the higher exploitation rates among widely used vendors but also provides a detailed timeline perspective. Such insights are critical for understanding how exploitation patterns evolve over time, thereby enabling better anticipation of future trends and targeted mitigation strategies.

#### 4.2.5.2 Comparative Analysis of Non-Exploited CVEs

To provide a more comprehensive view of the vulnerability landscape, we extended our analysis to examine non-exploited CVEs across top vendors. Figure 4.22 illustrates the annual distribution of non-exploited CVEs for the top 10 vendors from 1999 to 2024.

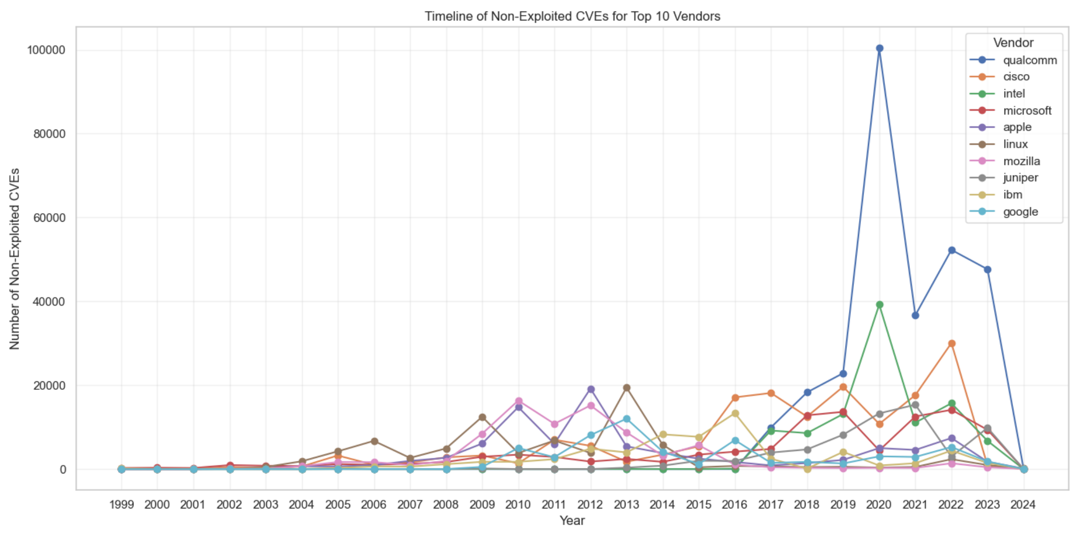


FIGURE 4.22: Timeline of non-exploited CVEs for the top 10 vendors. The line chart depicts annual non-exploited CVE counts, revealing significant patterns in vulnerability disclosure and exploitation potential.

**Insights from Non-Exploited CVEs Timeline Analysis** When comparing the patterns of non-exploited vulnerabilities with exploited ones, several revealing insights emerge:

- **Extraordinary Vulnerability Volumes:** Qualcomm exhibits an exceptional pattern with a dramatic spike in non-exploited CVEs during 2021, reaching approximately 100,000 vulnerabilitiesâĂŤa scale vastly exceeding all other vendors. This anomalous peak, followed by sustained high numbers in 2022-2023 (approximately 50,000 vulnerabilities), suggests massive vulnerability disclosure events potentially related to systematic security audits or architectural reviews rather than typical discovery patterns.

- **Secondary Volume Leaders:** Intel shows a significant peak of nearly 40,000 non-exploited CVEs in 2020, while Cisco reaches approximately 30,000 in 2022. These volumes, while substantial, remain significantly below Qualcomm's unprecedented levels.

- **Exploitation-to-Disclosure Ratio:** The comparison between exploited and non-exploited CVEs reveals that only a small fraction of disclosed vulnerabilities are actually exploited in the wild. For example, while Microsoft consistently shows high exploitation numbers (often exceeding 400 exploited CVEs annually), its non-exploited CVE counts remain relatively moderate compared to vendors like Qualcomm and Intel.

- **Temporal Patterns:** Prior to 2017, non-exploited vulnerability counts remained relatively modest across all vendors (generally below 20,000 annually). The dramatic increases observed from 2018 onward suggest fundamental shifts in vulnerability discovery, disclosure practices, or reporting methodologies rather than actual changes in software security.

- **Consistency vs. Volatility:** Some vendors (like Apple, Mozilla, and IBM) show relatively stable patterns of non-exploited CVEs over time, while others (particularly Qualcomm, Intel, and Cisco) demonstrate extreme volatility.

This suggests different approaches to vulnerability management and disclosure among major technology vendors.

The extreme disparity between exploited and non-exploited CVEs, particularly for vendors like Qualcomm, raises important questions about vulnerability prioritization. While the sheer volume of non-exploited vulnerabilities might suggest effective security controls preventing exploitation, it more likely reflects the reality that most vulnerabilities, despite being theoretically exploitable, lack the technical feasibility, attacker motivation, or accessibility required for practical exploitation in real-world scenarios. This comparative analysis provides security practitioners with crucial context for vulnerability prioritization, suggesting that raw vulnerability counts alone offer limited insight into actual security risk. Instead, exploitation likelihoodâĂŤinfluenced by factors such as ease of exploitation, attacker motivation, and security control effectivenessâĂŤmay serve as a more meaningful metric for security resource allocation.

### 4.2.6 Deep Dive into CVEs with High Exploits

To gain a deeper understanding of critical vulnerabilities, it is important to focus on CVEs with a high number of associated exploits as shown in Figure (4.16). This involves exploring details like descriptions and information from resources such as the Open Source Vulnerability Database (OSVDB).

#### 4.2.6.1 CVE with more than 13 exploits

For the sake of simplicity, we will analyze the CVEs with more than 13 exploits as shown in (Table 4.10).

| **CVE ID** | **Exploit Related** | | | | | **CVE Related** | | |
|---|---|---|---|---|---|---|---|---|
| | **Exploits** | **Exploit Date** | **Author** | **Platform** | **Type** | **CVSS Score** | **Reserved Date** | **CWE** |
| CVE-2004-0067 | 14 | 2004-01-12 | JeiAR | PHP | WebApps | 4.3 | 2004-01-14 | CWE-79 |
| CVE-2004-1925 | 15 | 2004-04-12 | JeiAR | PHP | WebApps | 7.5 | 2005-05-04 | CWE-89 |
| CVE-2006-4889 | 21 | 2006-09-12 | SHiKaA,ThE LeO | PHP | WebApps | 5.1 | 2006-09-19 | NA |
| CVE-2006-4985 | 22 | 2006-09-22 | HACKERS PAL | PHP | WebApps | 4.3 | 2006-09-25 | CWE-79 |
| CVE-2006-5190 | 17 | 2006-10-04 | lostmon | PHP | WebApps | 4.3 | 2006-10-06 | NA |
| CVE-2006-5911 | 38 | 2007-05-08 | anonymous | PHP | WebApps | 7.5 | 2006-11-15 | NA |
| CVE-2007-0182 | 28 | 2007-01-09 | IbnuSina | PHP | WebApps | 7.5 | 2007-01-10 | NA |
| CVE-2007-0900 | 16 | 2007-02-12 | K-159 | PHP | WebApps | 7.5 | 2007-02-13 | NA |
| CVE-2008-6543 | 36 | 2008-03-24 | ZoRLu | PHP | WebApps | 7.5 | 2009-03-29 | CWE-94 |
| CVE-2012-1199 | 38 | 2012-02-11 | indoushka | PHP | WebApps | 7.5 | 2012-02-17 | CWE-94 |
| CVE-2014-7910 | 14 | Multiple | Multiple | Multiple | Multiple | 7.5 | 2014-10-06 | NA |

TABLE 4.10: CVEs with their number of exploits, CVSS scores, and relevant dates.

We have total of 11 CVEs with more than 13 exploits. 10 CVEs are sharing the same characteristics, they target the same software using multi-entry points and/or files, except The **CVE-2014-7910** vulnerability, also known as "Shellshock," which is a high-severity security flaw in the GNU Bash (Bourne Again Shell). This vulnerability allows attackers to execute arbitrary commands on vulnerable systems by crafting malicious environment variables. In the appendix D, a summary of the 14 different exploits associated with **CVE-2014-7910**. These exploits are various methods for taking advantage of the "Shellshock" vulnerability in different environments and applications, ranging from web servers to hardware devices. Each exploit varies based on the platform it targets (e.g., CGI, Linux, hardware) and the way the payload is delivered (e.g., Metasploit modules, scripts). The common theme among these exploits is the manipulation of environment variables in Bash, which can trigger unintended command execution if the system is vulnerable to Shellshock. Proper patching and security measures, such as updating the Bash version to a patched one, are critical in mitigating these risks.

**By analyzing the remaining 10 CVEs**, for instance, the one with 38 exploits is interesting. CVE-2012-1199 has 38 associated exploits. Key columns for analysis will be all important features in ExploitDB except columns like *port*, *tags*, *aliases*, *screenshot_url*, and *application_url* which contain null values and can be disregarded for this analysis.

To analyze the variations among the exploits, we first group by different attributes such as description, author, type, and platform. Each record in the dataset represents a unique combination of these attributes. The common attributes for all records are:

- **Description**: All records describe vulnerabilities in "Basic Analysis and Security Engine (BASE) 1.4.5".
- **Author**: All records are authored by *"indoushka"*.
- **Type**: All records are classified as *"webapps"*.
- **Date added, date published, and date updated**: All records are added and published on the same date, 11th February 2012. However, 31 records were updated on 10th April 2015 and 7 records were updated on 14th April 2015.
- **Verified**: All exploits have been verified.
- **Platform**: All records target the *"php"* platform.
- **Codes**: All records share the same CVE *(CVE-2012-1199)* but have different OSVDB identifiers.

By analyzing the differences in the OSVDB identifiers within the codes column, we can understand variations in vulnerability details. The dataset contains a variety of OSVDB identifiers associated with **CVE-2012-1199**.

Since direct access to OSVDB is no longer available due to its shutdown in April 2016, we use MITRE CVE source map for OSVDB to check the differences in the OSVDB IDs mentioned above. However, the above-mentioned IDs are not available in the MITRE list, suggesting that they might have been deleted or merged. The numeric parts of the OSVDB IDs (ranging from 79511 to 79550) suggest they might be sequential, indicating entries created around the same time, potentially related to the same vulnerability but with slight variations.

Further analysis of the exploit descriptions reveals subtle differences in the text. Each description follows a similar structure, mentioning "Basic Analysis and Security Engine (BASE) 1.4.5" followed by a specific file path and the type of vulnerability, which is "Remote File Inclusion".

**Key Observations**:

1. **Common Pattern**:
    - All descriptions mention "Basic Analysis and Security Engine (BASE) 1.4.5" product.
    - The type of vulnerability mentioned in all descriptions is "Remote File Inclusion".

2. **Different File Paths**:
    - Each description points to a different file within the BASE application.
    - The file paths include files in various directories like /admin, /help, /includes, /setup, etc.
3. **Parameter for Inclusion**:
    - Most descriptions mention ?base_path as the parameter used for the file inclusion vulnerability.
    - There are a few exceptions with different parameters such as ?GLOBALS[user_session_path] and ?ado_inc_PHP.

Going into a detailed comparison and analysis of the descriptions of vulnerabilities in the Basic Analysis and Security Engine (BASE) 1.4.5 from the CVE database and Exploit DB as outlined in the appendix (A), we observe the following:

- The CVE description mentioned 42 entry points, while the exploit DB mentioned only 38 ones, however, all paths are matched regardless of the query ? used except one item (number 26 the appendix (A) admin/base_roleadmin.php).
- Both sources highlight the remote file inclusion vulnerabilities in various PHP files of BASE 1.4.5.
- The CVE database provides a more structured and exhaustive list of the affected files.
- Exploit DB descriptions corroborate the CVE database entries, confirming the vulnerabilities but with more details regarding the affected entry points.
- These vulnerabilities stem from improper handling of input parameters related to file inclusions.
- The risk is significant, as remote file inclusion can lead to the execution of arbitrary PHP code, compromising the server.

This analysis underscores the importance of addressing remote file inclusion vulnerabilities in BASE 1.4.5.

We have conducted an analysis of multiple CVEs with a high number of associated exploits, for instance, CVE-2008-6543, which has 36 exploits. Our findings reveal a consistent pattern: these CVEs exhibit multiple entries or techniques that can be employed to exploit one or more systems. This characteristic answers the question of **Why do these CVEs have multiple exploits**. More examples are in the appendix C which implies our analysis.

The observed pattern of a small number of CVEs attracting a disproportionately high number of exploits supports Allodi's [8] research on the heavy-tailed distribution in vulnerability exploitation. Our findings, particularly the 11 CVEs with more than 13 exploits each, exemplify his observation that certain vulnerabilities attract significantly more attention from attackers, creating exploitation hotspots that warrant special attention in security prioritization frameworks.

Both the CVE database and Exploit DB provide valuable insights into the scope and specifics of these vulnerabilities. Addressing these issues comprehensively can enhance the security posture of applications and prevent potential exploitation by malicious actors.

#### 4.2.6.2 CVE with 10 to 13 exploits

By using the same approach above, we have decided to go further to CVEs with 10 to 13 exploits, to find out if they share or have the common characteristics as above. The number of CVEs which have between 10 and 13 exploits are 23 CVEs. We analyzed it for consistency in author, type, platform, and description across their associated exploits, as well as similarity to the CVE description and the techniques used in exploitation.

**Out of the 23 CVEs analyzed:**

- **14 CVEs (60.87%)** show high consistency in their exploit characteristics and descriptions.
- **9 CVEs (39.13%)** show more variation, often due to:
  - Multiple authors contributing exploits
  - Different platforms being targeted
  - Varying approaches to exploiting the same vulnerability

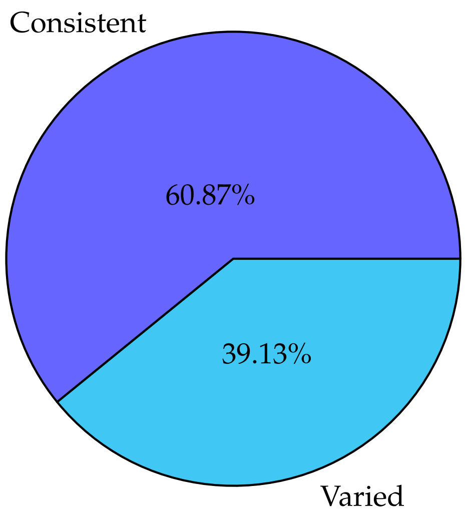


FIGURE 4.23: Distribution of CVE Consistency

This analysis reveals that a majority of the examined CVEs (60.87%) demonstrate high consistency in their associated exploits' characteristics and descriptions. This consistency suggests a focused approach to vulnerability exploitation, often by a single author or team targeting a specific platform or application.

However, a significant portion (39.13%) of the CVEs show more diversity in their exploits. This variability could indicate:

- **The consistency in characteristics is decreased when the number of exploits per CVE decreases as well**. More examples are in Appendix (C.2) which implies our analysis.
- Broader impact of the vulnerability across different systems or platforms
- Greater interest from the security community, resulting in multiple researchers developing exploits
- More complex vulnerabilities that can be exploited through various methods

These findings highlight the diverse nature of vulnerability exploitation and underscore the importance of comprehensive security strategies that account for potential variations in exploit characteristics, even within a single CVE.

This comprehensive analysis of multiple CVEs and their associated exploits reveals several key patterns and insights:

1. **Authorship Patterns:** Some CVEs have exploits from multiple authors, indicating widespread interest or a collaborative effort in the security community. Others have all exploits from a single author, suggesting focused research or specialized knowledge.

2. **Platform Diversity:** While many web application vulnerabilities are consistently targeted on PHP platforms, system-level vulnerabilities often have exploits across multiple operating systems, reflecting the diverse environments in which these vulnerabilities exist.

3. **Exploit Specificity:** Even within a single CVE, exploits often target different specific files, parameters, or system components. This highlights the complexity of many vulnerabilities and the multiple attack vectors they may present.

4. **Consistency in Exploit Types:** For most CVEs, the type of exploit (e.g., local, remote, web application) is consistent across all associated exploits, reflecting the nature of the underlying vulnerability.

5. **Evolution of Exploits:** In some cases, especially for more severe or widespread vulnerabilities, we see an evolution of exploits over time, with later exploits often being more sophisticated or targeting specific environments.

6. **Vendor Involvement:** In at least one case (CVE-2010-4120), we see the vendor (IBM) producing all the exploits, likely as part of their security research and patching process.

7. **Exploit Frameworks:** The recurring mention of Metasploit in several CVEs underscores the importance of exploit frameworks in vulnerability research and penetration testing.

These findings underscore the dynamic and diverse nature of exploit development in response to discovered vulnerabilities. They highlight the importance of comprehensive security strategies that account for multiple attack vectors, even when dealing with a single vulnerability. Furthermore, the analysis emphasizes the value of the security community's collaborative efforts in identifying and addressing vulnerabilities across diverse systems and platforms.

Key Insights from Exploit Analysis

**1. Exploit Trends and Verification:**

- Significant increase in exploit publications from 1988-2020, followed by a decline in 2021-2024
- Verified exploits (18,273) outnumber unverified ones (3,858), with verification most common for web applications (9,937 verified webapps exploits)
- The decline in recent years may indicate improved security practices, shifting attack patterns, or limitations in public reporting

**2. Exploitation Distribution Patterns:**

- 87.63% of exploited vulnerabilities (19,392 CVEs) have a single exploit, while 12.37% (2,738 CVEs) have multiple exploits
- Web application vulnerabilities dominate the exploit landscape (12,520 exploits), followed by remote (3,821) and denial of service (3,764) exploits
- Only 11 CVEs have more than 13 associated exploits, exemplifying Allodi's heavy-tailed distribution in vulnerability exploitation

**3. Vendor Exploitation Rates:**

- Microsoft shows the highest exploitation rate (9.13%) among major vendors, followed by Linux (7.65%) and Apple (6.37%)
- Enterprise-focused vendors experience higher exploitation rates compared to consumer-focused vendors
- Exploitation rates correlate with market presence and attack surface size, with widely deployed systems attracting more attention from attackers

**4. Multiple-Exploit Vulnerability Patterns:**

- CVEs with high exploit counts (10+ exploits) predominantly target web applications (60.87% show high consistency in author, platform, and techniques)
- Multi-exploit vulnerabilities often involve multiple entry points in the same application (e.g., remote file inclusion vulnerabilities in different files)
- The consistency in exploit characteristics decreases as the number of exploits per CVE decreases, suggesting more focused exploitation for highly targeted vulnerabilities

## 4.3 Patches

Patches represent the remediation component of vulnerability management, providing fixes for identified security issues. This section examines the characteristics and distribution of patches, focusing on Microsoft's approach to vulnerability remediation. While the temporal relationships between vulnerability disclosure, exploitation, and patching will be analyzed in depth in Chapter 5, this section focuses on the static characteristics of patches, including their distribution across product families, vulnerability types addressed, and related trends.

### 4.3.1 Microsoft Security Response Center (MSRC) CVRF

The Microsoft Security Response Center (MSRC) uses the Common Vulnerability Reporting Framework (CVRF) as a standardized framework to publish security updates and patches. The CVRF is released on a monthly basis, typically referred to as "Patch Tuesday," and may encompass multiple releases and updates within a single CVRF document. This approach ensures a comprehensive and coordinated update process to address various vulnerabilities.

To April 2024, Microsoft-related vulnerabilities account for a total of 19,620 CVEs. Of these, 6,335 CVEs (32%) were identified before 2016, while 13,285 CVEs (68%) emerged after 2016, based on CPE text matching with the CVE database. Since 2016, Microsoft has patched a total of 9,981 CVEs through its MSRC CVRFs, with 95 CVEs (1%) of these originating before 2016.

The total number of releases (CVRFs) are **96** which handled and remediated **9,981** CVEs. These reports have been systematically released since 2016, reflecting Microsoft's commitment to transparency and security.

### 4.3.2 Distribution of Patches Across Microsoft Product Families

Understanding which Microsoft product families receive the most patches provides valuable insights into the security landscape of Microsoft's ecosystem. Figure 4.25 presents the top 15 Microsoft product families by the number of patched vulnerabilities.

The creation of these subsets enables both broad pattern analysis across large vulnerability populations and focused examination of complete lifecycles where all events (disclosure, exploitation, and patching) are documented.

For the analysis of vulnerability distribution across Microsoft product families, we utilized data exclusively from the MSRC patched CVEs dataset (9,981 CVEs from 2016 to April 2024). This methodological choice was made because comprehensive product information was consistently available within the MSRC dataset, whereas when referencing the CPE information from the CVE-V5 database, we discovered that 3,054 of these patched CVEs (approximately 30.6%) had no associated CPE information.

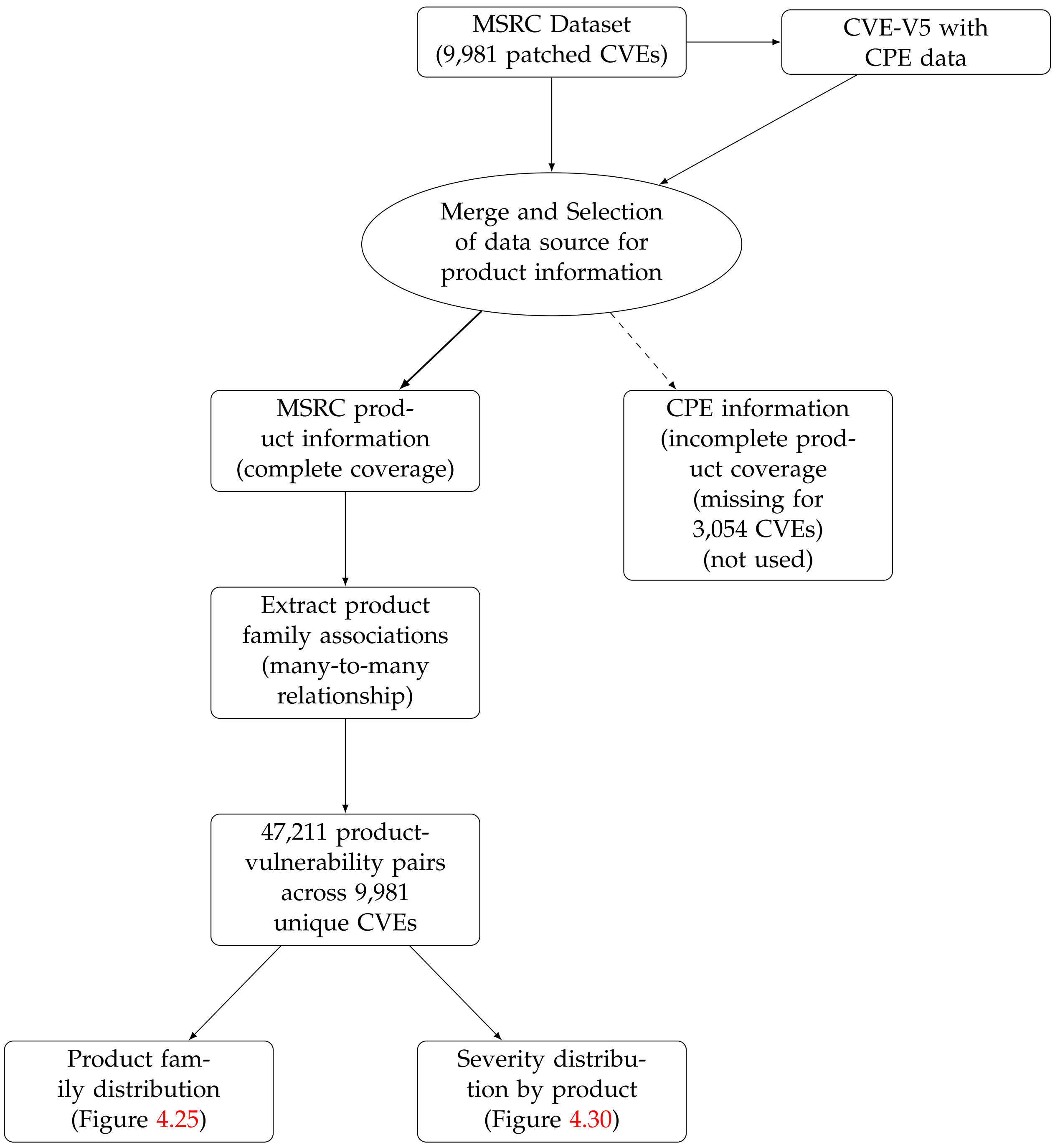

MSRC Dataset
(9,981 patched CVEs)
CVE-V5 with CPE data
Merge and Selection of data source for product information
MSRC product information (complete coverage)
CPE information (incomplete product coverage (missing for 3,054 CVEs) (not used)
Extract product family associations (many-to-many relationship)
47,211 product-vulnerability pairs across 9,981 unique CVEs
Product family distribution (Figure 4.25)
Severity distribution by product (Figure 4.30)

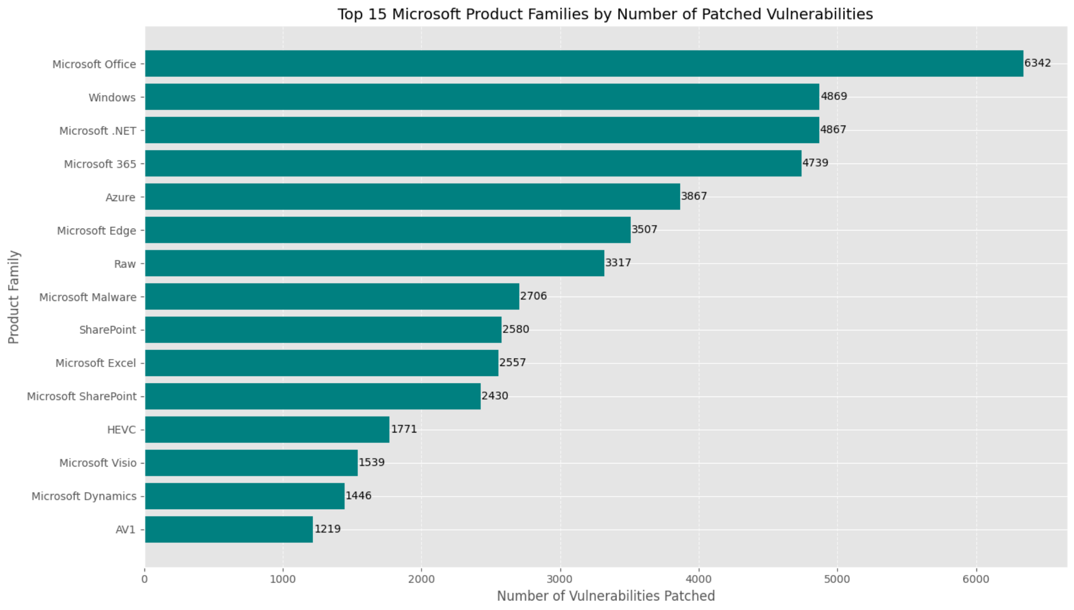
FIGURE 4.24: Data processing workflow for product family vulnerability analysis

The data processing workflow for this analysis is illustrated in Figure 4.24.

It is crucial to understand that the vulnerability counts presented in Figure 4.25 represent the number of patched vulnerabilities that affected each product family, not the number of unique CVEs. A single CVE may affect multiple Microsoft products or product families simultaneously. This many-to-many relationship between vulnerabilities and products explains why the cumulative count across all product families (47,211) significantly exceeds the total number of unique patched CVEs (9,981). For example, a single memory corruption vulnerability might affect both Windows and Office if they share a common component.

This same methodological consideration applies to the severity distribution analysis presented in Figure 4.30, where the vulnerability counts for each product family similarly reflect instances of affected products rather than unique CVEs.

FIGURE 4.25: Top 15 Microsoft Product Families by Number of Patched Vulnerabilities



For the analysis of vulnerability distribution across Microsoft product families, we utilized data exclusively from the MSRC patched CVEs dataset (9,981 CVEs from 2016 to April 2024). This methodological choice was made because comprehensive product information was consistently available within the MSRC dataset, whereas when referencing the CPE information from the CVE-V5 database, we discovered that 3,054 of these patched CVEs (approximately 30.6%) had no associated CPE information.

It is crucial to understand that the vulnerability counts presented in Figure 4.25 represent the number of patched vulnerabilities that affected each product family, not the number of unique CVEs. A single CVE may affect multiple Microsoft products or product families simultaneously. This many-to-many relationship between vulnerabilities and products explains why the cumulative count across all product families (47,211) significantly exceeds the total number of unique patched CVEs (9,981). For example, a single memory corruption vulnerability might affect both Windows and Office if they share a common component.

Microsoft Office leads with 6,342 patched vulnerabilities, significantly ahead of other product families. Windows and Microsoft .NET follow with 4,869 and 4,867 patched CVEs respectively, while Microsoft 365 accounts for 4,739 patched CVEs. This distribution reflects both the complexity and ubiquity of these products in enterprise environments, as well as their attractiveness as targets for threat actors.

The prominent position of Microsoft Office is particularly noteworthy, with nearly 30% more vulnerabilities than Windows. This may be attributed to Office's complex feature set, frequent updates, extensive legacy codebase, and its role as a common attack vector through document-based threats. Additionally, Office represents a family of products (Word, Excel, PowerPoint, etc.), each with its own attack surface.

Microsoft's cloud and web-oriented services also feature prominently, with Azure (3,867 patched CVEs) and Microsoft Edge (3,507 patched CVEs) ranking high in vulnerability counts. This reflects the growing importance of these platforms in Microsoft's product strategy and the security challenges inherent in cloud and browser technologies.

Enterprise collaboration tools are also significant sources of vulnerabilities, with SharePoint accounting for 2,580 patched CVEs and Microsoft SharePoint for an additional 2,430 patched CVEs. The distribution suggests a substantial concentration of security issues in collaboration platforms, which typically involve complex interactions between multiple components and users.

### 4.3.3 Common Weakness Types Addressed by Patches

Analyzing the types of weaknesses addressed by Microsoft patches provides insights into the nature of security issues most commonly remediated. Figure 4.27 illustrates the top 10 Common Weakness Enumerations (CWEs) addressed by Microsoft patches.

For the analysis of Common Weakness Enumeration (CWE) distribution in Microsoft patches, we merged the MSRC dataset with the comprehensive CVE-V5 database to obtain CWE information for patched vulnerabilities. It is important to note that 4,607 CVEs (approximately 46.2%) from the MSRC patched dataset had no associated CWE information in the CVE-V5 database. Our analysis therefore focuses on the remaining 5,374 CVEs that have identifiable CWE classifications.

Additionally, we should be aware that the frequency counts in Figure 4.27 may include duplications, as some CVEs are associated with multiple CWEs. This means that while we analyzed 5,374 unique CVEs with CWE information, the total count of CWE instances is higher due to these many-to-one relationships between CWEs and CVEs.

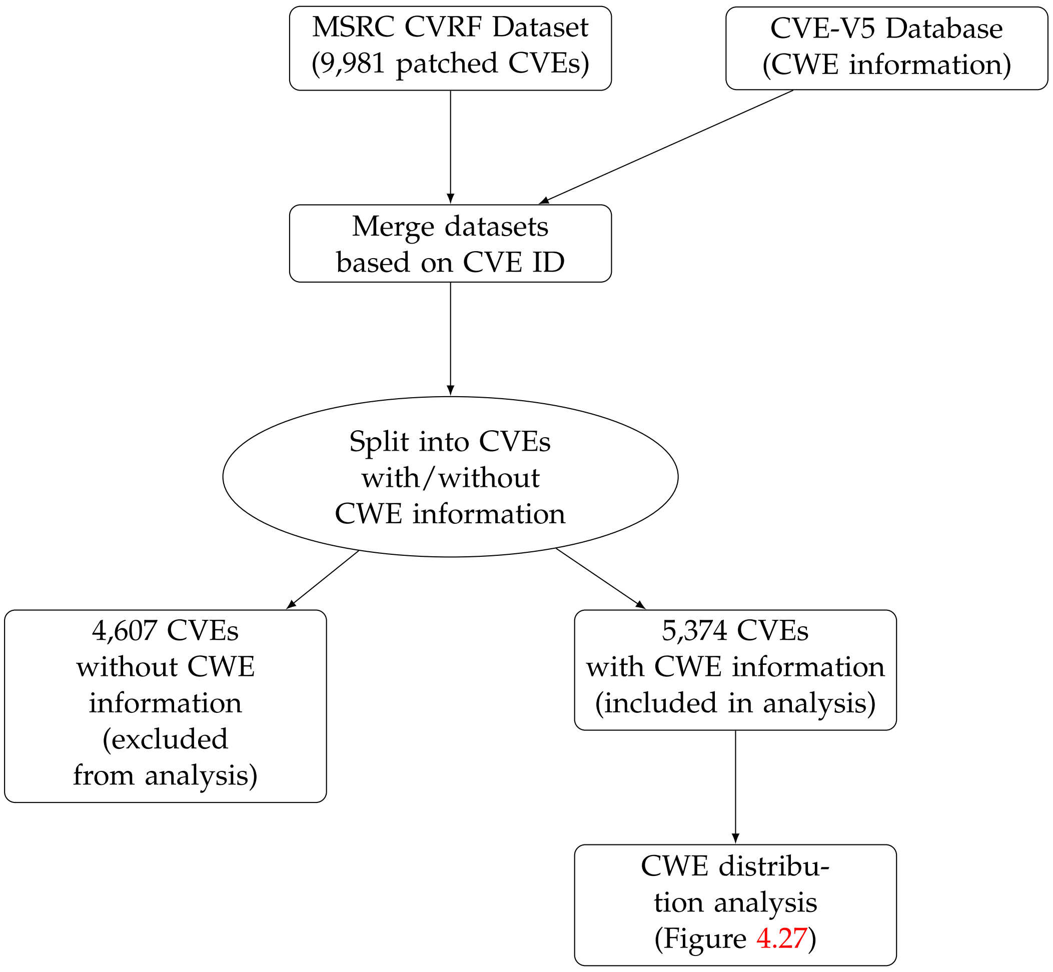

MSRC CVRF Dataset (9,981 patched CVEs)
CVE-V5 Database (CWE information)
Merge datasets based on CVE ID
Split into CVEs with/without CWE information
4,607 CVEs without CWE information (excluded from analysis)
5,374 CVEs with CWE information (included in analysis)
CWE distribution analysis (Figure 4.27)

FIGURE 4.26: Data processing workflow for CWE distribution analysis

The data processing workflow for this analysis is illustrated in Figure 4.26.

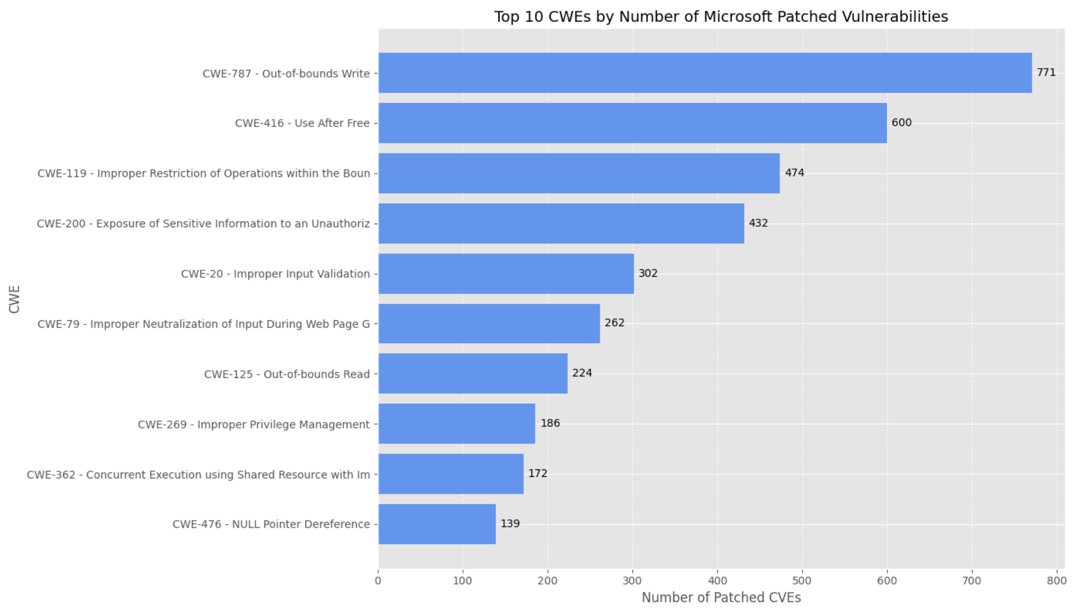


FIGURE 4.27: Top 10 CWEs by Number of Microsoft Patches

For the analysis of Common Weakness Enumeration (CWE) distribution in Microsoft patches, we merged the MSRC dataset with the comprehensive CVE-V5 database to obtain CWE information for patched vulnerabilities. It is important to note that 4,607 CVEs (approximately 46.2%) from the MSRC patched dataset had no associated CWE information in the CVE-V5 database. Our analysis therefore focuses on the remaining 5,374 CVEs that have identifiable CWE classifications.

Additionally, we should be aware that the frequency counts in Figure 4.27 may include duplications, as some CVEs are associated with multiple CWEs. This means that while we analyzed 5,374 unique CVEs with CWE information, the total count of CWE instances is higher due to these many-to-one relationships between CWEs and CVEs.

The most prevalent vulnerability weakness type addressed by Microsoft patches is CWE-787 (Out-of-bounds Write) with 771 CVEs patched, followed by CWE-416 (Use After Free) with 600 CVEs patched, and CWE-119 (Improper Restriction of Operations within the Bounds of a Memory Buffer) with 474 CVEs patched. This distribution reveals a significant concentration of memory safety issues among Microsoft products, accounting for a substantial portion of patched vulnerabilities. Other prominent vulnerability weakness types include CWE-200 (Exposure of Sensitive Information to an Unauthorized Actor) with 432 CVEs patched and CWE-20 (Improper Input Validation) with 302 CVEs patched.

The predominance of memory safety issues (CWE-787, CWE-416, CWE-119, CWE-125) among the top vulnerabilities is particularly noteworthy, exemplifying Anderson's [11] economic analysis of cybersecurity incentives. The prevalence of these issues supports his argument that fundamental architectural security improvements often face economic barriers, as addressing memory safety at its root requires substantial investment in redesigning core components, with benefits realized only over extended timeframes. Collectively, these memory-related weaknesses account for 2,069 patched CVEs, approximately 20.7% of all Microsoft patched CVEs analyzed.

This concentration of memory safety vulnerabilities aligns with Meunier's [102] observation that vulnerabilities often exist simultaneously at multiple abstraction levels, making them particularly resistant to comprehensive remediation. The persistence of these vulnerability types, despite awareness of their existence, reflects the "technical debt" phenomenon described by Akhoundali et al. [4], where foundational weaknesses continue to manifest despite efforts to address them.

Web-related vulnerabilities are also notable, with CWE-79 (Improper Neutralization of Input During Web Page Generation) accounting for 262 CVEs patched. This reflects the security challenges inherent in Microsoft's web-focused products and services. The prominence of privilege and authorization issues, represented by CWE-269 (Improper Privilege Management) with 186 CVEs patched, underscores the critical nature of access control in Microsoft's extensive product ecosystem, which often operates in enterprise environments with complex permission structures.

### 4.3.4 Evolution of CWE addressed by Patches

Further analysis of how the top CWEs have evolved over time provides additional context to Microsoft's security priorities and challenges. Figure 4.28 shows the yearly trends for the five most common CWEs in Microsoft patched vulnerabilities.

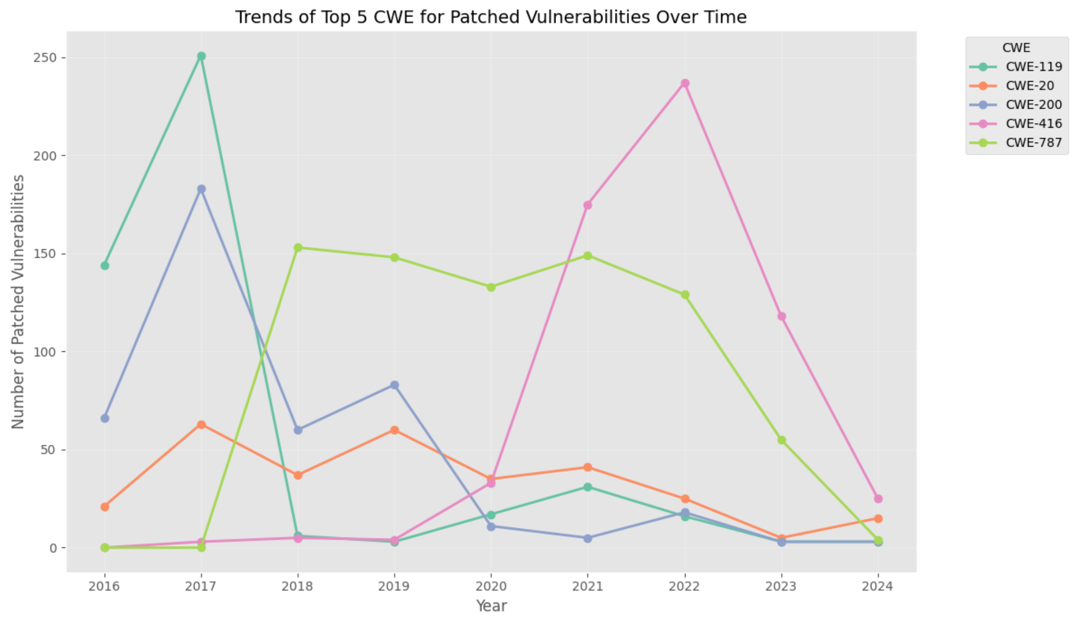


FIGURE 4.28: Trends of Top 5 CWE Patches Over Time

The temporal evolution of these top CWEs reveals several important patterns. Most notably, CWE-119 (Improper Restriction of Operations within Memory Buffer) showed an early peak in 2017 with approximately 250 patched CVEs, followed by a dramatic decline in subsequent years. This pattern suggests successful mitigation strategies or architectural changes that significantly reduced the occurrence of this particular vulnerability class over time.

CWE-416 (Use After Free) exhibited minimal presence from 2016 to 2020, but then showed a dramatic rise in 2021 and peaked in 2022 with approximately 240 patched CVEs, before declining in 2023 and 2024. This sudden emergence and subsequent decline suggests a focused period of identification and remediation for this specific vulnerability type, potentially due to increased scrutiny or detection capabilities during this timeframe.

CWE-787 (Out-of-bounds Write) showed a different pattern, emerging significantly in 2018 with about 150 patched CVEs and maintaining a relatively consistent presence through 2022, before declining in 2023. This sustained presence indicates a persistent challenge in addressing this type of memory safety issue over multiple years.

CWE-200 (Exposure of Sensitive Information) showed an early peak in 2017 with approximately 180 patched CVEs, followed by fluctuating but generally decreasing numbers in subsequent years. This suggests early identification and prioritization of information disclosure vulnerabilities, with continued monitoring and remediation.

CWE-20 (Improper Input Validation) shows moderate fluctuations throughout the analysis period, with peaks in 2017, 2019, and 2023. The recurrent nature of this vulnerability type underscores the persistent challenge of secure input handling in complex software systems.

These diverse temporal patterns indicate that different vulnerability classes follow distinct lifecycles, with some showing rapid emergence and resolution (CWE-119, CWE-416), while others persist consistently over time (CWE-787). The variations may reflect Microsoft's evolving security priorities, improvements in detection capabilities, changes in development practices, or shifts in attacker methodologies.

The decline observed across most CWE types in 2023-2024 aligns with the overall reduction in patches during that period, potentially indicating broader improvements in Microsoft's secure development lifecycle.

#### 4.3.4.1 Comparative Analysis: Microsoft vs. Overall Vulnerability Landscape

A comparative analysis between the distribution of CWEs in Microsoft patches and the overall distribution of CWEs across all CVEs provides valuable insights into how Microsoft's security priorities align with broader industry vulnerability trends. Table 4.11 presents this comparison for the top 10 CWEs in each category.

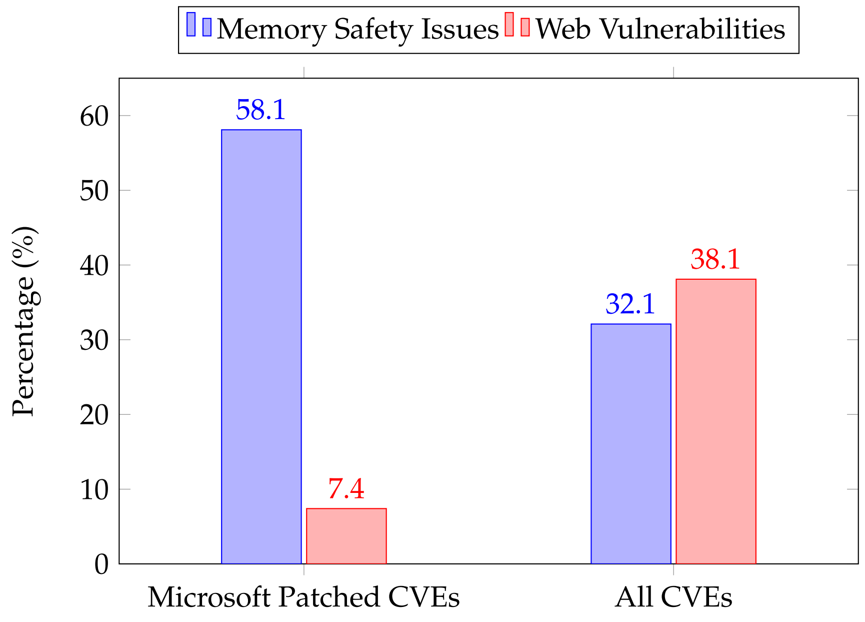
TABLE 4.11: Comparison of Top CWE Rankings: Microsoft Patches vs. All CVEs

| CWE | Weakness Type | MS Rank | MS Count | Overall Rank | Overall Count |
|---|---|---|---|---|---|
| CWE-787 | Out-of-bounds Write | 1 | 771 | 5 | 10,183 |
| CWE-416 | Use After Free | 2 | 600 | 10 | 4,141 |
| CWE-119 | Memory Buffer Restriction | 3 | 474 | 2 | 11,906 |
| CWE-200 | Information Exposure | 4 | 432 | 6 | 7,852 |
| CWE-20 | Improper Input Validation | 5 | 302 | 4 | 10,446 |
| CWE-79 | Cross-site Scripting | 6 | 262 | 1 | 26,728 |
| CWE-269 | Improper Privilege Management | 7 | 186 | - | - |
| CWE-125 | Out-of-bounds Read | 8 | 224 | 8 | 5,866 |
| CWE-362 | Concurrent Execution Issues | 9 | 172 | - | - |
| CWE-476 | NULL Pointer Dereference | 10 | 139 | - | - |
| CWE-89 | SQL Injection | - | - | 3 | 11,356 |
| CWE-22 | Path Traversal | - | - | 7 | 5,902 |
| CWE-352 | Cross-Site Request Forgery | - | - | 9 | 5,630 |

The analysis reveals significant differences in CWE prioritization between Microsoft patches and the general vulnerability ecosystem. Out of the top 10 CWEs addressed in Microsoft patches and the top 10 CWEs across all vulnerabilities, 7 CWEs appear in both lists (70% overlap), indicating substantial but not complete alignment between Microsoft's focus areas and the broader vulnerability landscape.

Microsoft places significantly higher emphasis on memory safety issues compared to the general vulnerability landscape. Memory-related weaknesses (CWE-787, CWE-416, CWE-119, CWE-125) account for 58.1% of Microsoft's top 10 patched vulnerabilities, compared to 32.1% across all CVEs as outlined in Figure 4.29. This pronounced focus highlights Microsoft's recognition of memory safety as a critical security concern in their products, particularly important given the prevalence of legacy code and system-level software in their ecosystem.

FIGURE 4.29: Comparative Distribution of Memory Safety vs. Web Vulnerabilities



Conversely, Microsoft's patches show considerably less focus on web application vulnerabilities compared to the industry average. Web vulnerabilities like Cross-site Scripting (CWE-79) and SQL Injection (CWE-89) represent only 7.4% of Microsoft's top 10 patched vulnerabilities, despite constituting 38.1% of vulnerabilities across all CVEs. Most notably, while CWE-79 (Cross-site Scripting) ranks as the most common weakness overall with 26,728 instances, it appears only sixth in Microsoft's patching priorities with 262 patches.

Microsoft's patch distribution includes three CWEs in its top 10 that do not appear in the overall top 10: Improper Privilege Management (CWE-269), Concurrent Execution Issues (CWE-362), and NULL Pointer Dereference (CWE-476). These Microsoft-specific priorities likely reflect the peculiarities of their software architecture and the critical nature of privilege management in enterprise environments where their products predominate.

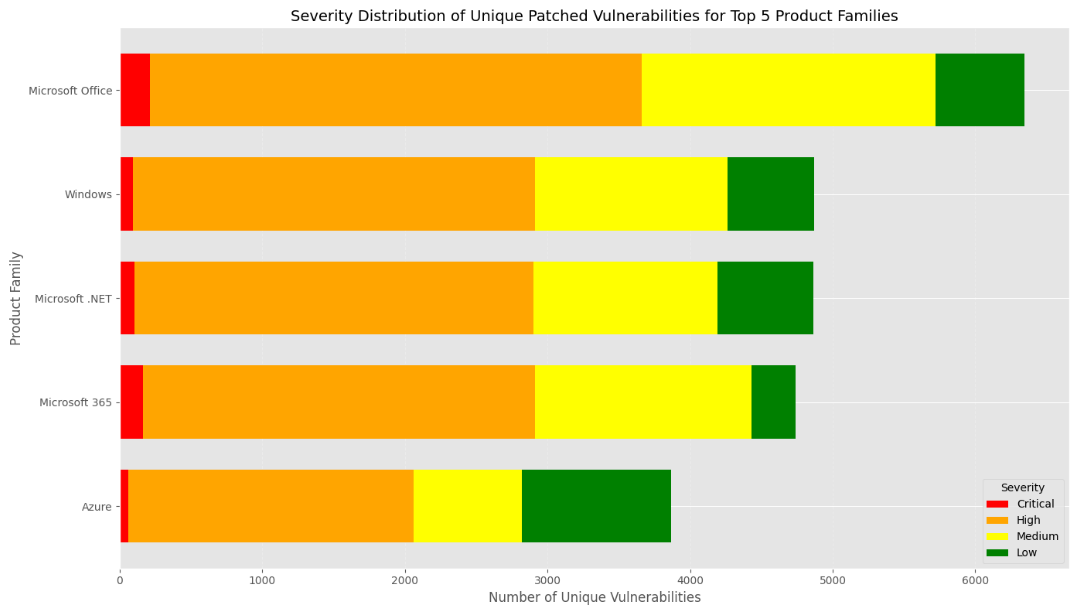
Significantly, SQL Injection (CWE-89), which ranks third in overall prevalence with 11,356 instances, does not appear in Microsoft's top 10 patched vulnerabilities. Similarly, Path Traversal (CWE-22) and Cross-Site Request Forgery (CWE-352) are common in the general ecosystem but absent from Microsoft's top patching priorities. This suggests either lower incidence of these vulnerability types in Microsoft products or potential gaps in addressing certain vulnerability classes.

These differences highlight Microsoft's targeted approach to vulnerability management, with an emphasis on issues that pose the highest risk to their specific product ecosystem. Microsoft's security priorities appear driven more by the potential impact of vulnerabilities in their specific software ecosystem than by the general prevalence of vulnerability types across the industry. This observation aligns with research by Tenable [150] regarding the importance of vendor-specific vulnerability analysis for accurate risk assessment.

### 4.3.5 Severity Distribution of Patched Vulnerabilities

The severity distribution of patched vulnerabilities offers insights into Microsoft's prioritization strategies and the overall risk landscape. Figure 4.30 illustrates the severity distribution across Microsoft's top product families.

FIGURE 4.30: Severity Distribution of Patched Vulnerabilities for Top 5 Product Families



This severity distribution analysis uses the same dataset described above for product family distribution, where a single CVE may affect multiple products simultaneously. The vulnerability counts for each product family reflect instances of affected products rather than unique CVEs.

The severity distribution reveals that High severity vulnerabilities constitute the largest proportion of patched security issues across all major product families. This pattern is particularly pronounced in Microsoft Office, Windows, Microsoft 365, and Microsoft .NET, where High severity vulnerabilities account for approximately 60-70% of all patched vulnerabilities.

Medium severity vulnerabilities represent the second most common category for Windows, Microsoft 365, and Microsoft .NET, while Low severity vulnerabilities take this position for Microsoft Office. This variance might reflect different risk profiles across these product families.

Azure exhibits a notably different distribution pattern, with a much higher proportion of Low severity vulnerabilities compared to other product families. This distinct profile could reflect Azure's cloud-native architecture, which potentially incorporates more modern security practices from inception.

Critical vulnerabilities, represented by the thin green segments at the left of each bar, constitute a very small fraction of the total vulnerabilities across all product families. Their relative rarity (less than 5% in all cases) suggests that Microsoft's secure development lifecycle is generally effective at preventing the most severe security issues.

### 4.3.6 Exploited CVEs in Microsoft and Differences with Exploit DB

In analyzing the relationship between vulnerabilities and their exploitation status, we identified significant discrepancies between the exploitation data reported by Microsoft and that available in the Exploit Database (Exploit DB).

According to Microsoft Security Response Center (MSRC) data, the number of CVEs with patches and associated exploits is relatively small, with 124 CVEs being reported as exploited. Conversely, Exploit DB, which is considered a comprehensive and unbiased source, lists 468 CVEs associated with Microsoft products that have been exploited. This significant discrepancy requires a thorough examination to ensure the accuracy and reliability of vulnerability exploitation analysis.

#### 4.3.6.1 Exploitation Reporting Statistics:

- **Total CVEs with Exploits Reported by Microsoft:**
  - Number of CVEs reported as exploited by Microsoft: 124
- **Exploited CVEs in Microsoft but Not Listed in Exploit DB:**
  - Number of CVEs exploited as per Microsoft but not present in Exploit DB: 105, such as *CVE-2018-8373*.
- **Exploited CVEs Listed in Exploit DB but Not Reported by Microsoft:**
  - Number of CVEs with records in Exploit DB: 468
  - Number of CVEs with records in Exploit DB and reported as exploited by Microsoft: 29
  - Number of CVEs not reported as exploited by Microsoft but present in Exploit DB: 439
- **Total Patched and Exploited CVEs:**
  - Total number of CVEs patched and exploited: 415
  - Number of patched CVEs not reported as exploited by Microsoft: 396, such as *CVE-2019-0612*
  - Number of patched and exploited CVEs according to both Microsoft and Exploit DB: 19

#### 4.3.6.2 Factors Contributing to Reporting Disparities:

Several factors may contribute to these significant differences in exploitation reporting:

- **Bias and Commercial Interests:** Microsoft may have commercial or reputational reasons to report lower numbers of exploited CVEs. Conversely, Exploit DB, as an independent entity, is likely free from such biases.
- **Multi-Vendor CVEs:** In many cases, a single CVE can affect multiple vendors. If an exploit targets a different vendor, Microsoft might not classify it as an exploited CVE, leading to discrepancies between the two data sets.
- **Update Frequency:** Microsoft's exploitation index is updated periodically and may not capture the latest exploitation incidents, whereas Exploit DB is updated continuously with contributions from the cybersecurity community.

These discrepancies highlight the importance of using multiple sources when analyzing vulnerability exploitation patterns. Relying solely on vendor-reported data may provide an incomplete picture of the true exploitation landscape. By cross-referencing vendor reports with independent databases like Exploit DB, security researchers and practitioners can develop a more comprehensive understanding of which vulnerabilities are being actively exploited in the wild.

Key Insights from Patch Analysis

**1. Product-Specific Vulnerability Concentrations:**

- Microsoft Office leads with 6,342 patched vulnerabilities, significantly ahead of Windows (4,869) and .NET (4,867), challenging assumptions that operating systems represent the most vulnerable components
- Enterprise products and services (SharePoint, Exchange, SQL Server) collectively account for a substantial portion of patched vulnerabilities, reflecting their complex attack surfaces
- Cloud and collaboration platforms (Azure, SharePoint) show growing vulnerability counts, highlighting the security challenges in rapidly evolving service architectures

**2. Weakness Type Distribution in Patches:**

- Memory safety issues (CWE-787, CWE-416, CWE-119, CWE-125) collectively account for 2,069 patched CVEs (20.7% of all analyzed CVEs), demonstrating persistent challenges in memory management
- Information disclosure (CWE-200) and input validation (CWE-20) vulnerabilities remain significant, highlighting ongoing challenges in secure data handling and input processing
- Privilege management issues (CWE-269) feature prominently, reflecting the complexity of access control in enterprise environments

**3. Temporal Evolution of CWE Patches:**

- Different vulnerability classes follow distinct lifecycles, with some showing rapid emergence and resolution (CWE-119, CWE-416) while others persist consistently (CWE-787)
- CWE-416 (Use After Free) showed a dramatic rise in 2021-2022 after minimal presence in previous years, suggesting focused identification and remediation efforts
- The decline across most CWE types in 2023-2024 may indicate improvements in Microsoft's secure development practices

**4. Exploitation Reporting Discrepancies:**

- Significant discrepancy between Microsoft-reported exploited CVEs (124) and Exploit DB records (468) highlights the importance of cross-referencing multiple sources
- Only 19 CVEs are reported as exploited in both sources, with 439 CVEs in Exploit DB not acknowledged as exploited by Microsoft
- These disparities underscore potential limitations in vendor transparency and the value of independent vulnerability tracking

The analysis of Microsoft's patch distribution provides valuable insights into the security landscape of their product ecosystem. The findings reveal that vulnerabilities are not evenly distributed across products, with Office, Windows, and .NET accounting for the largest shares. Memory safety issues represent a significant challenge, constituting over 20% of patched vulnerabilities. The temporal evolution of different vulnerability types shows distinct lifecycle patterns, suggesting evolving security priorities and improvements in certain areas.

The significant disparities in exploitation reporting between Microsoft and independent sources underscore the importance of using multiple data sources when analyzing vulnerability exploitation patterns. By combining insights from vendor reports and independent databases, security practitioners can develop a more comprehensive understanding of the true exploitation landscape.

The detailed temporal analysis of patching timelines, including the relationships between vulnerability disclosure, exploitation, and patch release, will be explored in Chapter 5. This upcoming analysis will provide deeper insights into the efficiency of Microsoft's vulnerability management processes and the evolving dynamics between attackers and defenders in the security landscape.

## 4.4 Summary

This chapter has provided a comprehensive exploratory data analysis of Common Vulnerabilities and Exposures (CVEs) and their associated exploits. Through the examination of over 246,000 CVE entries and 46,494 exploit records, we have uncovered significant patterns and trends that illuminate the current state of the cybersecurity landscape.

Our analysis reveals a dramatic increase in vulnerability disclosures over time, with a particularly striking 78% increase during the pandemic period (2020-2021). This acceleration exceeds the consistent 10% annual growth predicted by industry reports, suggesting that extraordinary circumstances can substantially impact vulnerability discovery and reporting rates. The distribution of vulnerabilities across severity levels shows a predominance of medium-severity issues (50.9%), challenging assumptions about severity inflation in recent years.

The persistence of specific weakness types, particularly Cross-site Scripting (CWE-79) which accounts for 15.35% of all mapped weaknesses, demonstrates Meunier's [102] observation that vulnerabilities often exist simultaneously at multiple abstraction levels, making them particularly resistant to comprehensive remediation. Our analysis of recurring CWEs across multiple years reflects the "technical debt" phenomenon described by Akhoundali et al. [4], where foundational weaknesses continue to manifest despite awareness of their existence.

The examination of vulnerability complexity through CWE relationships reveals that while 93% of CVEs have only one associated CWE, the remaining 7% exhibit more complex patterns with multiple weaknesses. Memory-related vulnerabilities dominate these multi-CWE relationships, with Out-of-bounds Write (CWE-787) appearing in 5 of the top 10 CWE pairs and 6 of the top 7 CWE triplets. This indicates a persistent challenge in addressing memory safety issues in software development.

Our analysis of exploits shows that web application vulnerabilities constitute the most prevalent exploit type, with 9,937 verified and 2,583 unverified exploits in this category. The distribution of exploits per CVE follows a heavy-tailed pattern, with 87.63% of exploited vulnerabilities having a single exploit and only 11 CVEs having more than 13 associated exploits. This pattern supports Allodi's [8] research on the heavy-tailed distribution in vulnerability exploitation, where certain vulnerabilities attract disproportionate attention from attackers.

Vendor-specific analysis reveals that Microsoft shows the highest exploitation rate among major vendors at 9.13%, followed by Linux (7.65%) and Apple (6.37%). This suggests that vendors with larger attack surfaces and enterprise focus experience higher exploitation rates, a finding that has significant implications for security prioritization and resource allocation.

The insights gained from this exploratory analysis provide crucial context for the temporal lifecycle analysis presented in Chapter 5, where we examine the dynamic relationships between vulnerability disclosure, exploitation, and patching over time. By establishing the static characteristics and distributions of vulnerabilities and exploits, this chapter lays the foundation for understanding their complex interactions throughout their lifecycles, ultimately contributing to more effective security strategies and vulnerability management practices.

# CHAPTER 5

# Lifecycle Analysis

The lifecycle of vulnerabilities represents a critical dimension in cybersecurity research, encompassing the sequence and timing of key events from vulnerability creation through discovery, disclosure, exploitation, and remediation. Understanding these temporal patterns provides essential insights into the effectiveness of security practices and the dynamics between attackers and defenders. This chapter presents a methodical examination of vulnerability lifecycles from 2016 to 2024, with particular focus on Microsoft-related vulnerabilities.

Building upon Frei's seminal work on vulnerability lifecycles [46], which established the foundational framework for understanding the temporal dynamics between vulnerability events, this analysis extends his methodology with contemporary data and refined statistical approaches. Frei's model identified five distinct phases: creation, discovery, disclosure, exploitation, and patch availability. Our research focuses particularly on the measurable relationships between disclosure (as evidenced by CVE reservation), exploitation (documented in exploit databases), and patching (through vendor security bulletins).

The investigation addresses several key research questions:

- What are the typical timeframes between vulnerability disclosure and exploitation, and how have these patterns evolved over time?
- How do patch development and deployment timelines relate to exploitation patterns?
- To what extent does vulnerability severity influence the timing of exploitation and patching?
- What factors contribute to outlier cases where vulnerabilities remain unpatched for extended periods?

The analysis integrates multiple datasets to provide comprehensive coverage of vulnerability lifecycles. These include:

- The CVE Version 5 database, which provides standardized data on vulnerability disclosure dates
- The Exploit Database (ExploitDB), documenting public exploits and their publication dates
- Microsoft Security Response Center (MSRC) data, containing patch release information for Microsoft-related vulnerabilities

By examining the temporal relationships between these events, this chapter offers insights into vulnerability management practices, the effectiveness of coordinated disclosure processes, and the challenges of timely remediation. The findings have significant implications for understanding the dynamics of the "race" between attackers and defenders, while providing an empirical foundation for improving security response strategies.

The chapter begins with a framework for temporal analysis, followed by two-point analyses examining CVE-Exploit and CVE-Patch relationships independently. It then progresses to a more complex three-point analysis integrating all lifecycle events across the 415 Microsoft-related vulnerabilities that have been both exploited and patched. Special attention is given to outlier cases and severity-based differences, concluding with implications for security practices and recommendations for vulnerability management.

## 5.1 Temporal Analysis Framework

The methodological approach to vulnerability lifecycle analysis requires careful consideration of data sources, temporal metrics, and analytical techniques. This section outlines the framework employed to examine the relationships between key vulnerability lifecycle events.

### 5.1.1 Lifecycle Events Data Considerations

For this analysis, we focused on vulnerabilities disclosed between 2016 and 2024, using the earliest documented date for each lifecycle event (CVE reservation date, exploit publication date, and patch release date). For each vulnerability $v$, we identified the time of disclosure ($t_{discl}(v)$), which serves as the reference point for subsequent temporal analyses. This approach aligns with methodology established by Frei [46] while extending it to accommodate contemporary data structures and sources.

The data used includes non-duplicated exploit records and CVEs from the CVE Version 5 database, along with patch information from Microsoft's Security Response Center (MSRC). As shown in Figure 5.1, which depicts the distribution of vulnerability lifecycle events for Microsoft from 2016 to 2024, these sources provide comprehensive coverage of the key temporal events in vulnerability management.

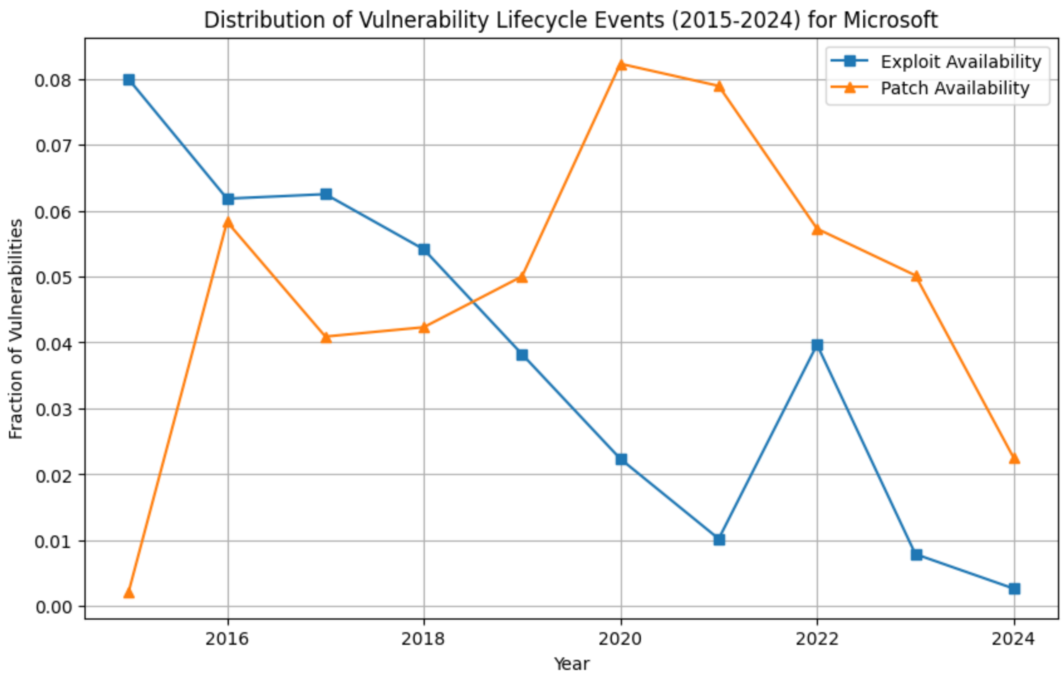


FIGURE 5.1: Distribution of Vulnerability Lifecycle Events (2016-2024) for Microsoft.

The methodological approach of analyzing vulnerability lifecycle events from 2016 to 2024 extends Frei's [46] foundational framework by incorporating a larger, more contemporary dataset. While Frei established key terminology and relationships between lifecycle events, our temporal analysis provides an updated view of these dynamics in the modern security landscape. The use of the disclosure time ($t_{discl}(v)$) as a reference point aligns with Dissanayake's [37] findings on the significance of disclosure timing in vulnerability management workflows.

### 5.1.2 Temporal Metrics and Statistical Approach

To quantify the timing relationships between lifecycle events, we calculated several key metrics:

- **Time to Exploit ($t_{exploit} - t_{discl}$)**: The time difference between the disclosure of a vulnerability (typically CVE reservation) and the publication of an exploit. Positive values indicate the exploit was published after disclosure, while negative values indicate pre-disclosure exploitation.
- **Time to Patch ($t_{patch} - t_{discl}$)**: The time difference between disclosure and the release of a patch addressing the vulnerability. This metric provides insight into vendor response times.
- **Exploit-Patch Gap ($t_{patch} - t_{exploit}$)**: The time difference between exploit publication and patch availability. Negative values indicate that exploitation occurred before patching, representing a critical window of vulnerability.

For each metric, we conducted comprehensive statistical analysis, calculating central tendency measures (mean, median, mode) to identify typical patterns, along with dispersion measures (range, standard deviation) to quantify variability. The use of multiple statistical indicators provides a more nuanced understanding of these temporal relationships, addressing limitations in previous studies that often relied solely on averages.

### 5.1.3 Dataset Integration and Subsets

The analysis integrates multiple data sources to create a comprehensive view of vulnerability lifecycles. From the full dataset of 246,422 CVEs and 46,494 exploit records, we extracted the following key subsets for detailed analysis:

- **Exploit Timeline Analysis**: Examining the temporal relationship between CVE reservations and exploit publications for all vulnerabilities in the ExploitDB, consisting of 22,130 unique CVEs with associated exploits.
- **Microsoft Patch Analysis**: Analyzing the patching patterns for 9,981 Microsoft-related CVEs patched since 2016, regardless of exploitation status.
- **Complete Lifecycle Analysis**: Focusing on 415 Microsoft-related CVEs that have been both exploited and patched, enabling comprehensive analysis of the entire vulnerability lifecycle. Within this subset, 391 CVEs are directly related to Microsoft products, while 24 pertain to third-party software that operates on Microsoft platforms.

### 5.1.4 Severity-Based Analysis

Recognizing that vulnerability severity significantly influences management practices, we stratified our analysis by severity levels (Critical, High, Medium, Low) based on standardized CVSS scores. This stratification enables identification of severity-specific patterns in exploitation and patching timelines, providing insights into how organizations prioritize vulnerability management based on potential impact.

This approach builds upon research by Costa et al. [31], who identified limitations in current severity scoring systems for predicting exploitation patterns. By examining temporal relationships across different severity levels, we can evaluate the effectiveness of severity-based prioritization strategies and identify potential misalignments between theoretical risk levels and actual exploitation patterns.

### 5.1.5 Outlier Analysis

While central tendency measures provide valuable insights into typical patterns, outlier analysis reveals important edge cases in vulnerability management. We employed specific thresholds (such as lifecycle differences exceeding 1,000 days) to identify and analyze outlier cases. This approach aligns with Allodi's research on heavy-tailed distributions in vulnerability exploitation [8], which suggests that a small subset of vulnerabilities exhibits significantly different characteristics from the broader population.

For each identified outlier, we conducted detailed investigation to understand contributing factors, particularly focusing on differences between Microsoft-specific vulnerabilities and those in third-party components. This methodical approach to outlier analysis provides important context for understanding the challenges in managing complex vulnerability landscapes.

The temporal analysis framework described here provides a structured approach to examining vulnerability lifecycles, enabling both broad pattern identification and focused investigation of specific vulnerability subsets and cases. The following sections present the results of applying this framework to the datasets described, beginning with two-point temporal analyses (CVE-Exploit and CVE-Patch) before proceeding to the more complex three-point lifecycle analysis.

## 5.2 Two-Point Temporal Analysis

This section examines the temporal relationships between pairs of vulnerability lifecycle events, focusing first on the time between CVE reservation and exploit publication, and then on the gap between CVE reservation and patch availability. This two-point analysis provides foundational insights into exploitation patterns and patching efficiency before proceeding to more complex multi-point lifecycle examinations.

### 5.2.1 CVE and Exploit Timeline Analysis

The temporal relationship between CVE reservation dates and exploit publication represents a critical dimension in understanding vulnerability lifecycle patterns. Analysis of exploit timing data from 1999 through 2024 reveals significant evolution in vulnerability discovery and disclosure practices, reflecting the maturation of industry security protocols and coordination mechanisms.

#### 5.2.1.1 Historical Evolution of Exploitation Patterns

Historical data analysis reveals distinct patterns in exploit timing relative to CVE reservations, as illustrated in Figures 5.2 and 5.3. In the early period (1999-2005), the security landscape was characterized by a predominance of pre-CVE exploits, with 315 out of 334 exploits in 1999 being published before their corresponding CVE reservations. This pattern suggests a reactive approach to vulnerability documentation during the industry's early years.

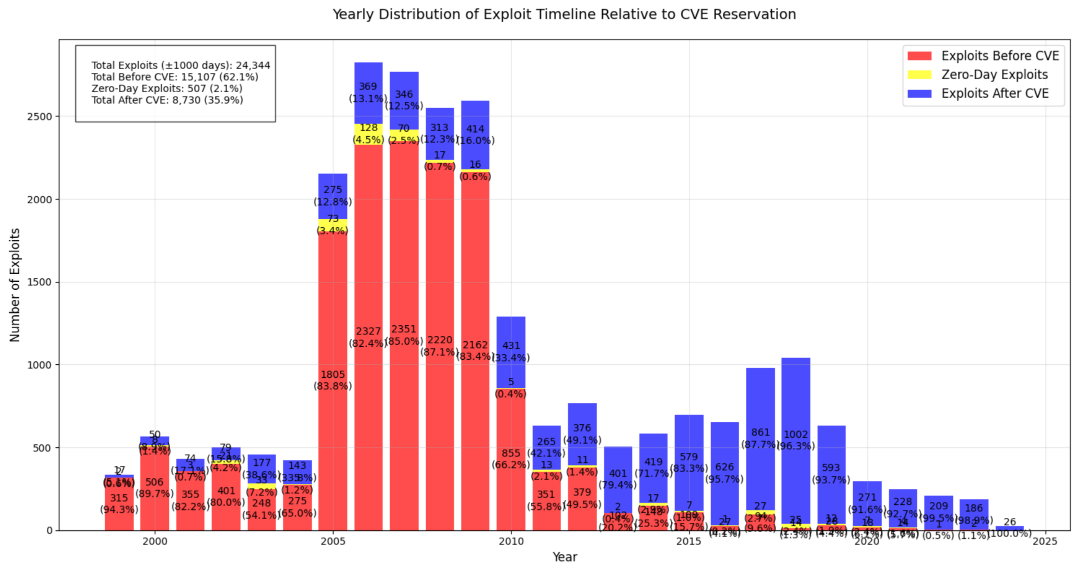


FIGURE 5.2: Distribution of exploit publication timing relative to CVE reservation (1999-2024). The stacked bars represent pre-CVE exploits (red), zero-day exploits (yellow), and post-CVE exploits (blue).

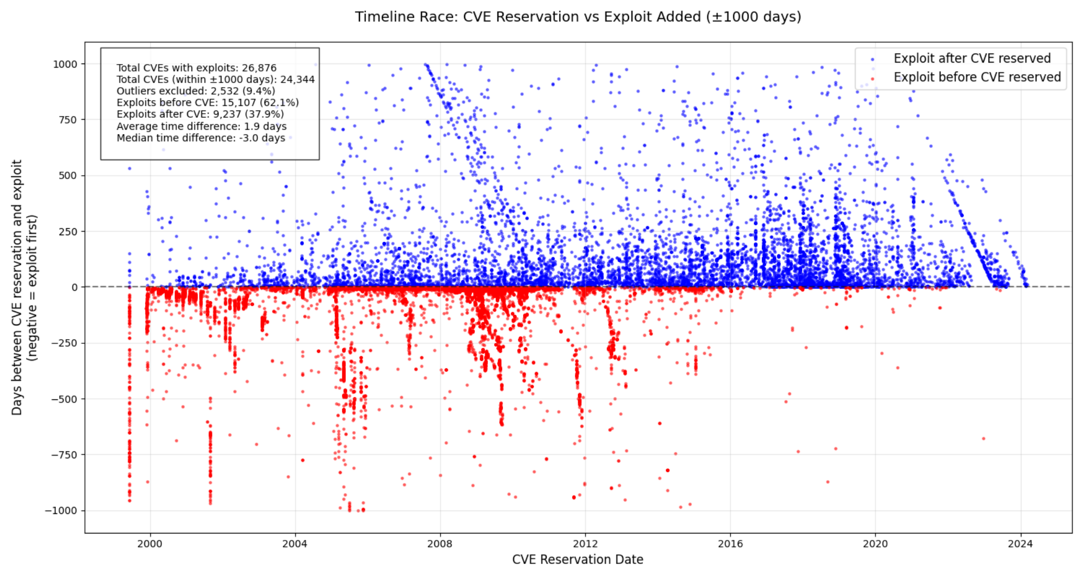


FIGURE 5.3: CVE exploitation race pattern (1999-2024).

The transition period around 2012 marked a significant shift, showing an approximately equal distribution between pre- and post-CVE exploits, indicating evolving security practices and improved coordination mechanisms. This evolution mirrors the advancement of coordinated vulnerability disclosure practices advocated by major vendors and security organizations during this period, aligning with Pastor et al.'s [117] assertion that improved information sharing would lead to more structured vulnerability management.

Contemporary data (2016-2024) shows a marked improvement in coordination between discovery and disclosure, with an average delay of 168 days between CVE reservation and exploit publication. This trend becomes even more pronounced in recent years, with 2022 showing 209 out of 210 exploits appearing after CVE reservation, demonstrating significantly improved industry coordination. The most recent data from 2024 Q1 further reinforces this trend, with all 26 documented exploits following CVE reservation, averaging 41.1 days delay.

The observed evolution from predominantly pre-CVE exploits before 2012 to a post-2012 landscape dominated by post-CVE exploitation ($\geq$ 90% in 2020-2024) indicates substantial improvements in vulnerability management practices. This transformation aligns with Ruohonen's [130] observations regarding increasing temporal variations in vulnerability reporting, though our analysis reveals a more dramatic improvement than previously documented.

#### 5.2.1.2 Zero-Day Exploitation Patterns

Zero-day exploits represent a particularly significant category, with 507 confirmed cases identified across the study period. These instances, where exploit publication coincides with CVE reservation, demonstrate the ongoing challenges in vulnerability management despite improved practices. The data reveals varying patterns in zero-day exploitation across different platforms and years, with notable clusters observed between 2003 and 2019.

The persistence of zero-day exploits, albeit at a relatively low rate (never exceeding 7.2% of annual cases), challenges some assumptions in existing literature about their prevalence. However, it aligns with Rajasooriya et al.'s [123] observation that most vulnerabilities follow predictable exploitation patterns after disclosure.

The relatively low percentage of zero-day exploits in our dataset contrasts with the heightened concern these vulnerabilities receive in security literature, suggesting a potential misalignment between perceived and actual risks in this area.

#### 5.2.1.3 Detailed Exploitation Timeframes

Detailed analysis of exploitation timeframes reveals diverse patterns: quick exploits emerging within 0-7 days of CVE reservation (particularly common in 2018-2020), pre-CVE exploits typically appearing 1-7 days before reservation, and long-delay cases extending up to 881 days post-reservation. These variations highlight the complex nature of exploitation dynamics and the multiple factors influencing exploit development timing.

The shift in exploitation patterns over time reflects significant changes in the security landscape. The early predominance of pre-CVE exploits suggests either limited coordination in vulnerability reporting or a security community less focused on responsible disclosure practices. The gradual transition to post-CVE exploitation indicates improved industry coordination, better communication between researchers and vendors, and more mature vulnerability management processes. This evolution contradicts the "milk or wine" hypothesis proposed by Ozment and Schechter [114], which suggested that older code tends to have fewer undiscovered vulnerabilities. Instead, our data indicates that improvements in security processes, rather than code maturity alone, play a significant role in vulnerability management outcomes.

#### 5.2.1.4 Platform-Specific Exploitation Patterns

Microsoft-specific temporal relationships between CVE reservations and exploit publications reveal distinctive patterns that differ notably from industry-wide trends. Figure 5.4 illustrates the evolution in Microsoft's vulnerability discovery and disclosure ecosystem from 1999 to 2023.

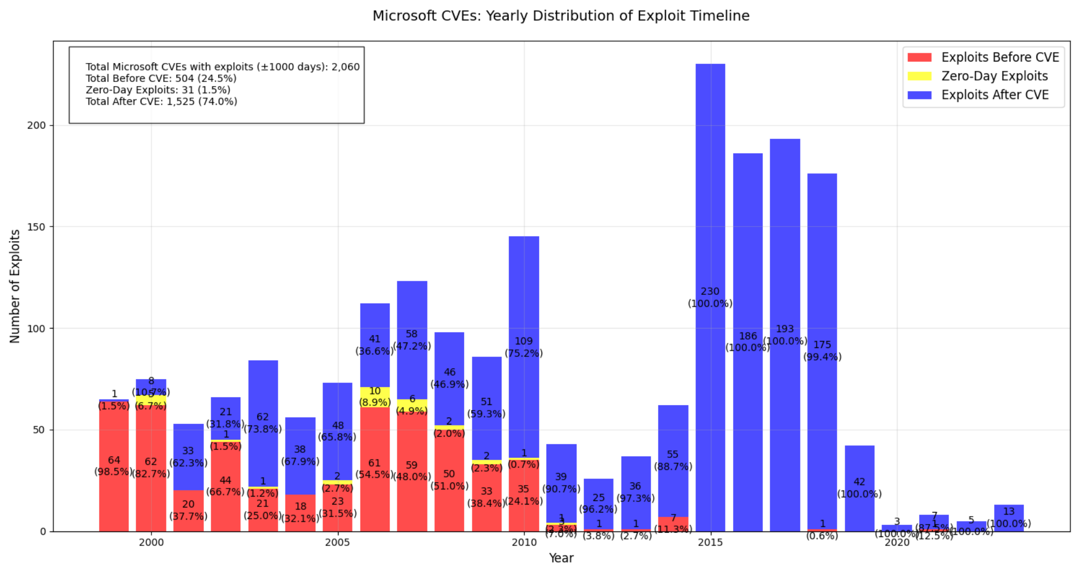


FIGURE 5.4: Distribution of Microsoft-specific exploit timing relative to CVE reservation (1999-2023). The visualization demonstrates the evolution from predominantly pre-CVE exploits (red) to post-CVE exploits (blue), with zero-day exploits (yellow) showing periodic spikes.

During the initial period (1999-2000), Microsoft's vulnerability landscape was characterized by a striking 98.5% of exploits being published before their corresponding CVE reservations, reflecting the reactive security posture common to that era. This pattern underwent a dramatic transformation, particularly from 2015 onwards, where nearly all exploit publications followed CVE reservations, indicating Microsoft's successful implementation of coordinated vulnerability disclosure practices.

The peak activity period for Microsoft vulnerabilities occurred during 2015-2017, with annual volumes reaching approximately 200 CVEs, presenting a more concentrated pattern compared to broader industry trends. Zero-day exploits in Microsoft products have shown a distinct historical trajectory, with notable peaks in 2006 (10 cases, representing 8.9% of that year's total) and 2000 (5 cases, 6.7%). The virtual absence of documented zero-day exploits since 2011 suggests either enhanced security practices or evolved disclosure processes within Microsoft's ecosystem.

This transformation aligns with Microsoft's introduction of structured vulnerability management practices, including the establishment of regular patch cycles and improved coordination with security researchers. The concentrated nature of Microsoft's exploitation patterns, compared to industry-wide data, reflects the company's systematic approach to vulnerability management and its influential role in shaping industry security practices. This pattern supports findings from Shahzad et al. [134], who documented Microsoft's increasingly structured approach to vulnerability management.

Key Insights from CVE-Exploit Timeline Analysis

**1. Dramatic Evolution in Exploit Timing:**

- A significant shift from 80%+ pre-CVE exploits in 1999-2004 to 80%+ post-CVE exploits in 2015-2024, indicating fundamental improvements in vulnerability disclosure and coordination practices.
- Microsoft's even more pronounced transformation from 98.5% pre-CVE exploits in 1999-2000 to almost exclusively post-CVE exploits after 2015.

**2. Zero-Day Exploitation Remains Limited:**

- Zero-day exploits (published same day as CVE reservation) have consistently remained below 7.2% of annual exploitation cases.
- For Microsoft products, documented zero-day exploits have become increasingly rare since 2011, suggesting improved security practices or disclosure processes.

**3. Exploit Patterns Reflect Security Practice Evolution:**

- The transition period around 2012 marked an inflection point in industry-wide exploitation patterns.
- Contemporary exploitation typically occurs 168 days after CVE reservation, allowing significant time for remediation when proper vulnerability management processes are in place.

**4. Microsoft's Distinct Exploitation Profile:**

- Microsoft's vulnerability landscape shows a more concentrated and structured pattern than the broader industry, reflecting its systematic approach to vulnerability management.
- The peak exploitation period for Microsoft (2015-2017) coincided with substantial changes in its security response framework.

### 5.2.2 CVE and Patch Timeline Analysis

The temporal relationship between vulnerability disclosure and patch availability provides crucial insights into vendor response capabilities and patch management efficiency. This section examines the time between CVE reservation and patch release, with particular focus on Microsoft's patching practices from 2016 to 2024.

#### 5.2.2.1 Temporal Distribution of Patching Activity

Examining the distribution of Microsoft patches over time reveals significant patterns and trends in their vulnerability management strategy. As illustrated in Figure 5.5, there has been a remarkable evolution in Microsoft's patching activity since 2016.

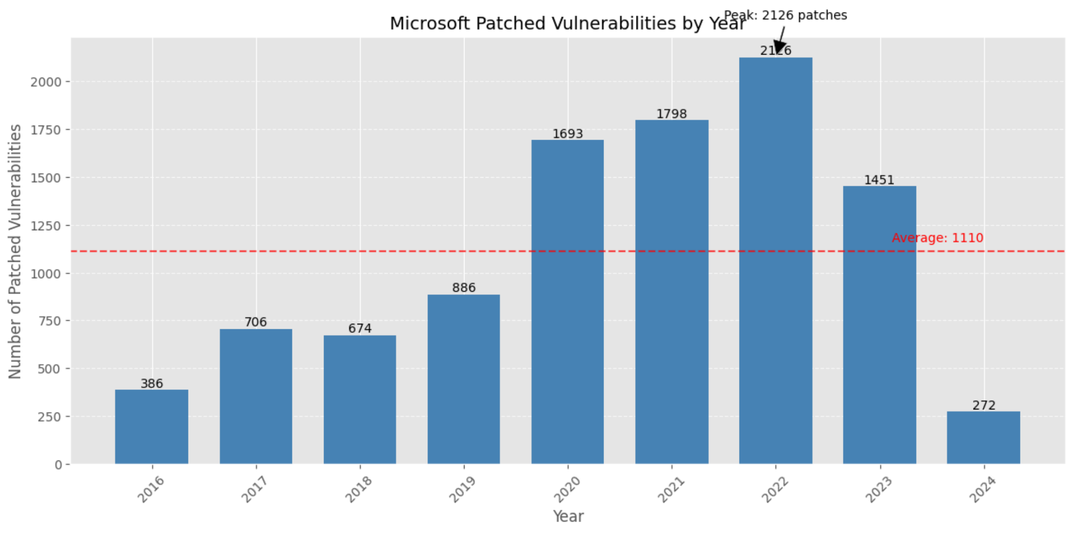


FIGURE 5.5: Microsoft Patched Vulnerabilities by Year (2016-2024).

The temporal analysis demonstrates a substantial increase in patching activity from 386 patched CVEs in 2016 to a peak of 2,126 patched CVEs in 2022. This illustrates the growing challenges of vulnerability management described by Johnson et al. [70]. This 451% increase aligns with their research on the complexities of timely patch deployment in enterprise environments, where increasing software complexity and interconnectedness create exponentially more potential vulnerability points requiring remediation. The average number of patched vulnerabilities annually stands at 1,110, with notable fluctuations observed throughout this timeframe.

Between 2016 and 2019, a steady increase in patching activity was evident, but 2020 marked a significant acceleration with 1,693 patched CVEs. This notable rise coincides with the onset of the COVID-19 pandemic, which dramatically altered work environments globally and potentially expanded attack surfaces for Microsoft products as remote work became prevalent. This observation extends findings from Lallie et al. [83], who documented increased cybersecurity vulnerabilities during the pandemic period.

The unprecedented peak observed in 2022 (2,126 CVEs patched) reflects several possible factors: an intensified focus on security by Microsoft, improved vulnerability detection mechanisms, increased reporting by security researchers, or a genuine increase in vulnerabilities within Microsoft products as they grow in complexity. This substantial increase also suggests a maturation of Microsoft's security response capabilities, enabling them to address vulnerabilities at scale.

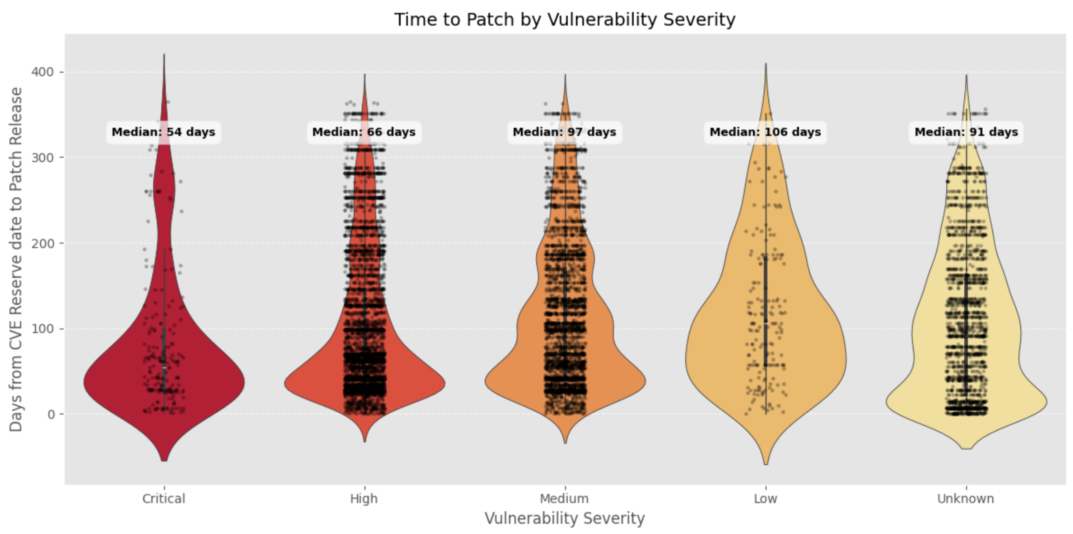
The year 2023 showed a decline to 1,451 patched CVEs, representing a 32% decrease from the 2022 peak. This reduction could indicate either the success of earlier remediation efforts and secure development practices, or potentially a strategic shift in Microsoft's approach to vulnerability management. The first quarter of 2024 shows a continuation of this downward trend with 272 patched CVEs, though this figure is expected to increase throughout the remainder of the year.

#### 5.2.2.2 Patching Response Time by Severity

The time between CVE reservation and patch release for Microsoft patched CVEs provides valuable insights into Microsoft's prioritization and response capabilities.

Figure 5.6 illustrates the distribution of patching times across different severity levels.

FIGURE 5.6: Time to Patch by Vulnerability Severity



The analysis reveals that Critical vulnerabilities show the shortest median patching time at 54 days, followed by High severity vulnerabilities at 66 days. Medium and Low severity vulnerabilities exhibit longer median response times at 97 and 106 days respectively, while vulnerabilities with Unknown severity have a median patching time of 91 days.

This severity-aligned prioritization pattern aligns with security best practices, where higher-risk vulnerabilities receive more immediate attention and resources. The significantly shorter response time for Critical vulnerabilities (54 days) compared to Low severity issues (106 days) represents approximately a 49% reduction in patching time, highlighting Microsoft's risk-based approach to vulnerability remediation. This pattern supports assertions by Xiong et al. [162] regarding the importance of severity-based prioritization, though their research also highlighted limitations in current severity scoring systems.

The violin plot widths indicate the distribution variability within each severity category. Critical vulnerabilities show a narrower distribution, suggesting more consistent and standardized response processes for these high-priority issues. In contrast, High, Medium, and Low severity vulnerabilities display wider distributions, indicating greater variability in response times possibly due to their lower prioritization or greater diversity in complexity.

These observations align with Dissanayake's [37] research on Microsoft's security response processes, which found that severity ratings significantly influence patch development priorities. However, the substantial median patching times even for Critical vulnerabilities (54 days) highlight the persistent challenges in vulnerability remediation described by Xiong et al. [162], who noted that patch development processes often involve complex trade-offs between security, functionality, and resource constraints.

Key Insights from CVE-Patch Timeline Analysis

**1. Severity-Based Prioritization:**

- Microsoft demonstrates clear severity-based prioritization in patch development, with Critical vulnerabilities patched in a median of 54 days compared to 106 days for Low severity issues (49% faster response).
- Narrower distribution of patching times for Critical vulnerabilities suggests more standardized remediation processes for high-priority issues.

**2. Pandemic Impact on Patching Volumes:**

- Significant acceleration in patching activity occurred in 2020 (1,693 patched CVEs), coinciding with the COVID-19 pandemic and widespread shift to remote work.
- This pattern extends Lallie et al.'s [83] findings on increased cybersecurity vulnerabilities during the pandemic period.

**3. Substantial Growth in Vulnerability Management Scale:**

- 451% increase in patching activity from 2016 (386 CVEs) to 2022 (2,126 CVEs) illustrates the growing complexity of vulnerability management.
- This dramatic growth reflects the expansion of attack surfaces as software systems become more complex and interconnected.

**4. Recent Decline in Patching Volume:**

- 32% decrease in patched CVEs from 2022 to 2023 (2,126 to 1,451) may indicate improved secure development practices or strategic shifts in vulnerability management.
- This decline follows a period of substantial patching activity, suggesting potential maturation in Microsoft's security development lifecycle.

The two-point temporal analyses presented in this section reveal distinct patterns in both exploitation and patching timelines. The evolution from predominantly pre-CVE exploits to primarily post-CVE exploits demonstrates improved security coordination across the industry, while Microsoft's patch distribution patterns reflect both their prioritization strategies and the specific security challenges within their product ecosystem. These findings provide the foundation for the more complex three-point analysis in the following section, which integrates exploitation and patching data to examine complete vulnerability lifecycles.

## 5.3 Three-Point Lifecycle Analysis

Building upon the two-point temporal analyses, this section examines the complete vulnerability lifecycle by integrating all three key events: CVE reservation, exploit publication, and patch release. This comprehensive analysis provides deeper insights into the complex interplay between vulnerability disclosure, exploitation, and remediation processes.

### 5.3.1 Integrated CVE-Exploit-Patch Lifecycle

The integrated analysis focuses on a subset of 415 Microsoft-related CVEs that have documentation for all three lifecycle events. This cohort represents vulnerabilities with complete lifecycle information, enabling detailed examination of the relationships between disclosure, exploitation, and patching. Of these 415 CVEs, 391 (94%) are directly related to Microsoft products, while 24 (6%) pertain to third-party software that operates on Microsoft platforms.

#### 5.3.1.1 CVE to Exploit Temporal Relationship

The analysis of time differences between CVE reservation and exploit publication reveals important patterns in the vulnerability exploitation timeline. Figure 5.7 illustrates this relationship for CVEs directly related to Microsoft products.

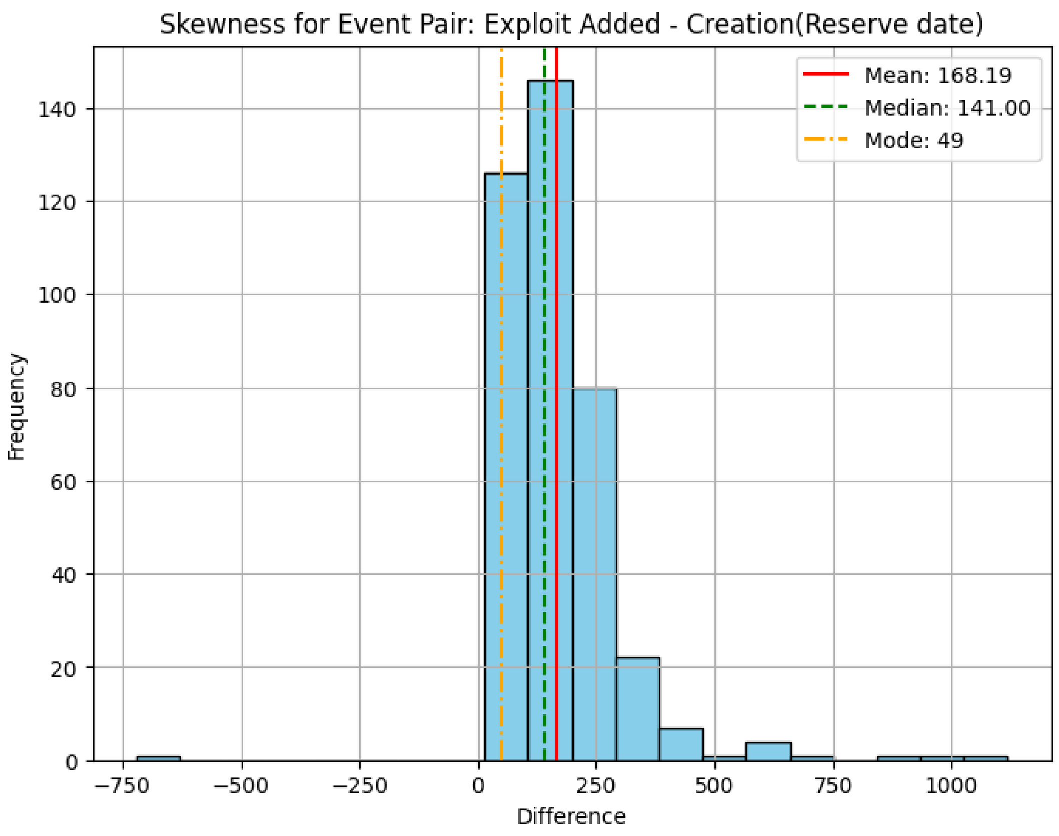


FIGURE 5.7: Distribution of differences in days between Exploit Adding date and CVE Creation date for Microsoft products.

For Microsoft-related CVEs, the analysis reveals that, on average, it takes about 168 days for a vulnerability to be exploited after its CVE is reserved (mean value). However, there is significant variation, with the median at 141 days, indicating that many vulnerabilities are exploited sooner. The most frequent time gap (mode) between reservation and exploitation is 49 days, suggesting that a considerable number of vulnerabilities are exploited relatively quickly. The distribution shows a positive skewness of 1.35, indicating an extended tail of delayed exploitations.

In extreme cases, it can take over three years for an exploit to appear, while some exploits are recorded before the CVE is officially reserved, as indicated by negative values in the distribution. These pre-CVE exploits likely result from delays in reporting or backdating of CVEs for vulnerabilities discovered through active exploitation. The complete range spans from -800 to 1118 days, highlighting the substantial variability in exploitation timelines.

When third-party applications operating on Microsoft platforms are included in the analysis (Figure 5.8), the patterns remain broadly similar, with a mean of 165 days and unchanged median of 141 days. This consistency suggests that exploitation patterns for vulnerabilities in the Microsoft ecosystem follow similar dynamics regardless of whether the vulnerable component is developed by Microsoft or a third party.

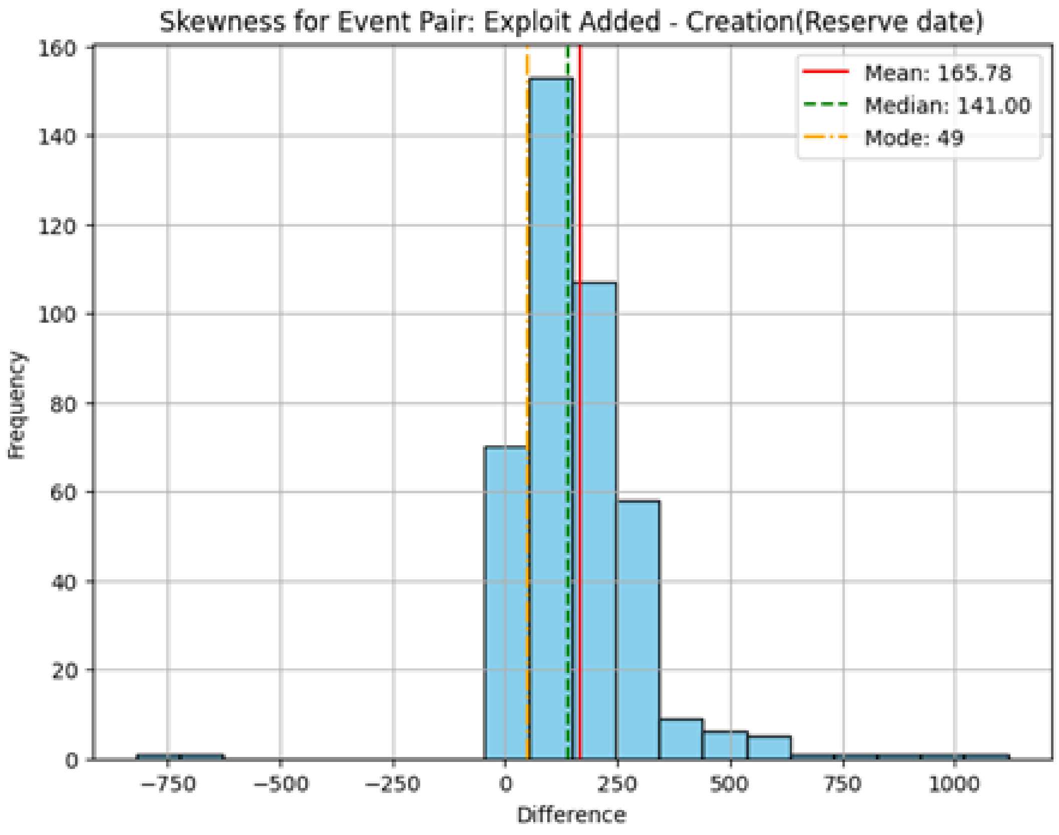


FIGURE 5.8: Distribution of differences in days between Exploit Adding date and CVE Creation date (including third-party applications).

The observed mean difference of 168 days between CVE reservation and exploitation for Microsoft products extends research by Dissanayake [37], who identified significant delays in vulnerability disclosure processes. However, our finding in Section 5.3.2 that critical vulnerabilities exhibit longer exploitation timelines (median 192 days) than medium-severity ones (median 138 days) challenges conventional assumptions in risk assessment frameworks. This counterintuitive result aligns with Ruohonen's [130] observation that critical vulnerabilities often face longer assessment periods, creating an extended exposure window for the most severe security issues.

#### 5.3.1.2 Exploit to Patch Temporal Relationship

The temporal relationship between exploit publication and patch availability represents a critical security dimension, as it defines the window during which systems remain vulnerable after exploitation becomes possible. Figure 5.9 shows this relationship for Microsoft products.

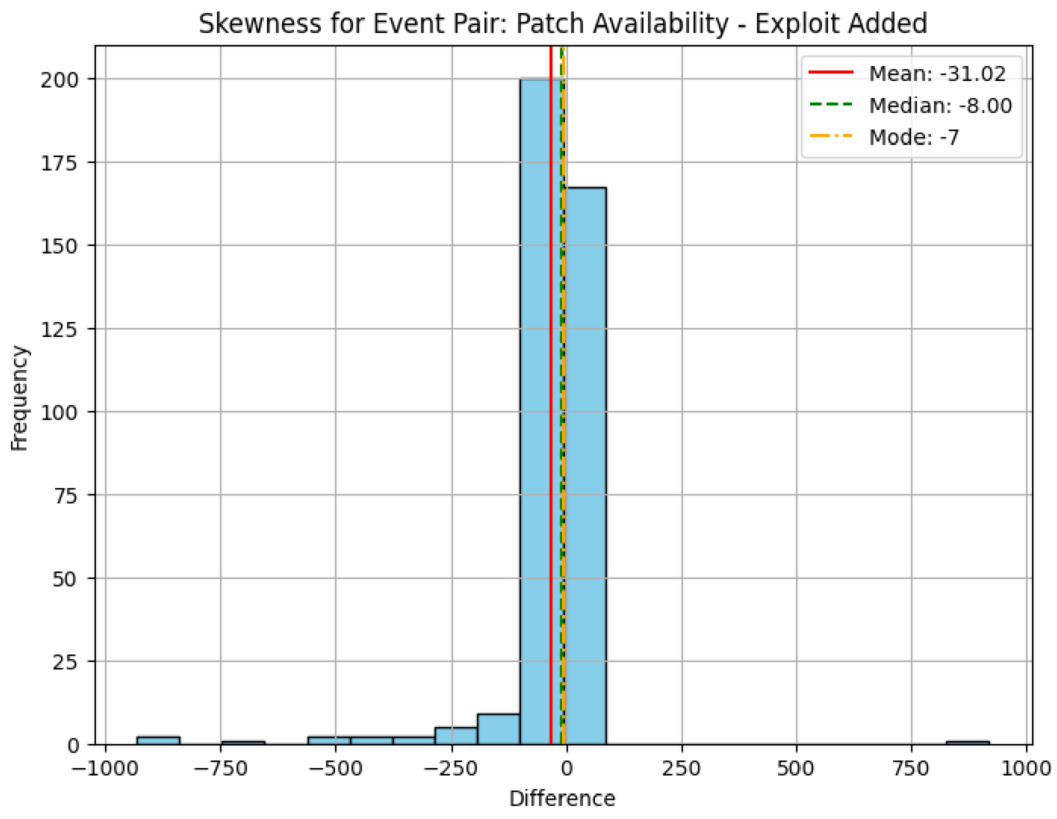


FIGURE 5.9: Distribution of differences in days between Patch Availability and Exploit Adding dates for Microsoft products.

The analysis indicates that exploits tend to be published around a month before patches become available, with a mean difference of -31 days. The median shows that in half of the cases, exploits are observed within a week before patch availability (-7 days), and the most common scenario (mode) is that exploits are added seven days prior to patching. This demonstrates a clear pattern where vulnerabilities are often actively exploited before patches are released, leaving systems vulnerable during this critical period.

The distribution shows substantial negative skewness (-3.74), emphasizing that most exploits occur before patches, though a few instances show patches being released long after exploitation. The range extends from -932 days (patch released almost three years after exploit publication) to 1,013 days (patch preceding exploit by nearly three years), with most values clustered in the negative range.

When third-party applications are included (Figure 5.10), the distribution shows greater variability and a positive mean of 59 days, contrasting with the negative mean for Microsoft-specific products. However, the median remains negative at -8 days, suggesting that the typical case still involves exploitation preceding patching. The extreme skewness (8.93) in this combined dataset points to the presence of significant outliers, particularly in third-party applications where patches may be substantially delayed.

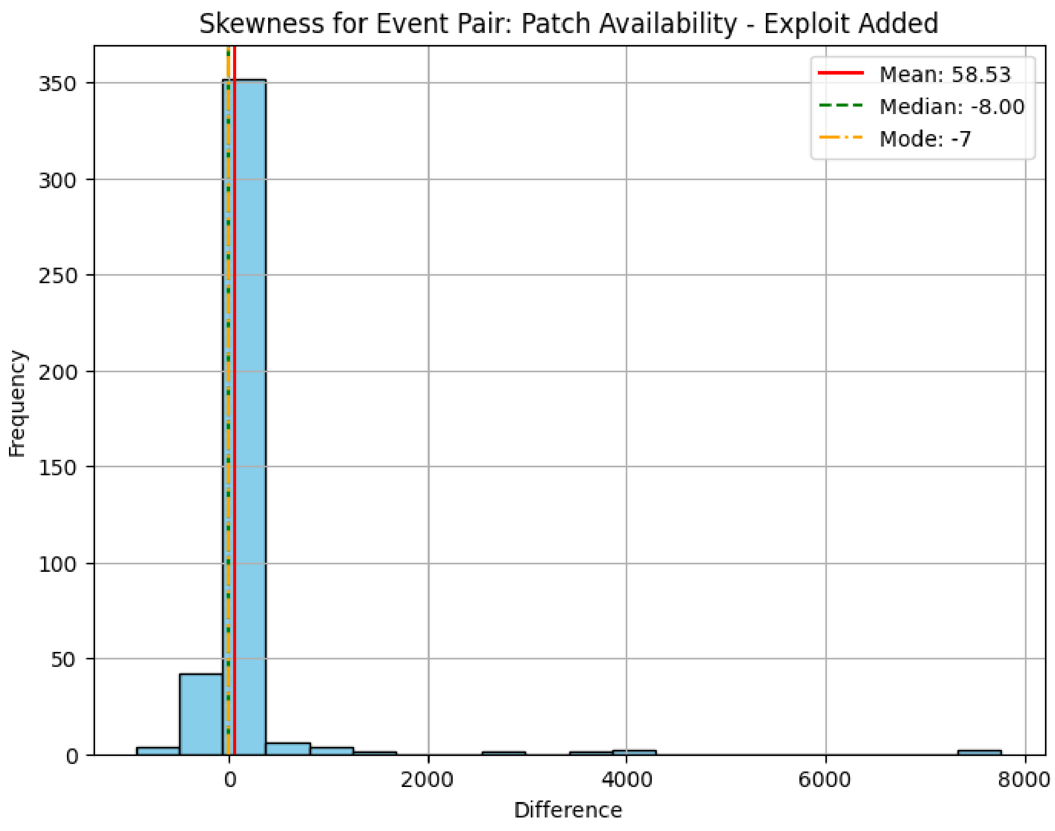


FIGURE 5.10: Distribution of differences in days between Patch Availability and Exploit Adding dates (including third-party applications).

The negative median difference between patch availability and exploit addition (-8 days for all Microsoft-related CVEs) indicates a prevailing pattern where exploits are typically published before patches become available. This "exploit-patch gap" supports findings by Woo et al. [161], who identified persistent challenges in patch development and deployment timelines. However, the mean difference of 59 days (when including third-party applications) suggests significant variability, with some patches being released long after exploitation. This variability aligns with Xiong et al.'s [162] research on the complex relationship between vulnerability disclosure and patch development, while extending their work by quantifying the typical gap between exploitation and patching in a major vendor's ecosystem.

#### 5.3.1.3 Complete Lifecycle Analysis

The complete lifecycle analysis examines the entire vulnerability progression from CVE reservation through exploitation to patching. This integrated view provides insights into the overall security response timeline and the interplay between disclosure, exploitation, and remediation processes.

Figure 5.11 visualizes the timeline of exploits and patches for the 415 Microsoft-related CVEs with complete lifecycle information. This visualization helps identify trends in the "patch vs. exploit race" by plotting the gap between exploit and patch dates, with CVEs arranged chronologically based on their reservation dates.

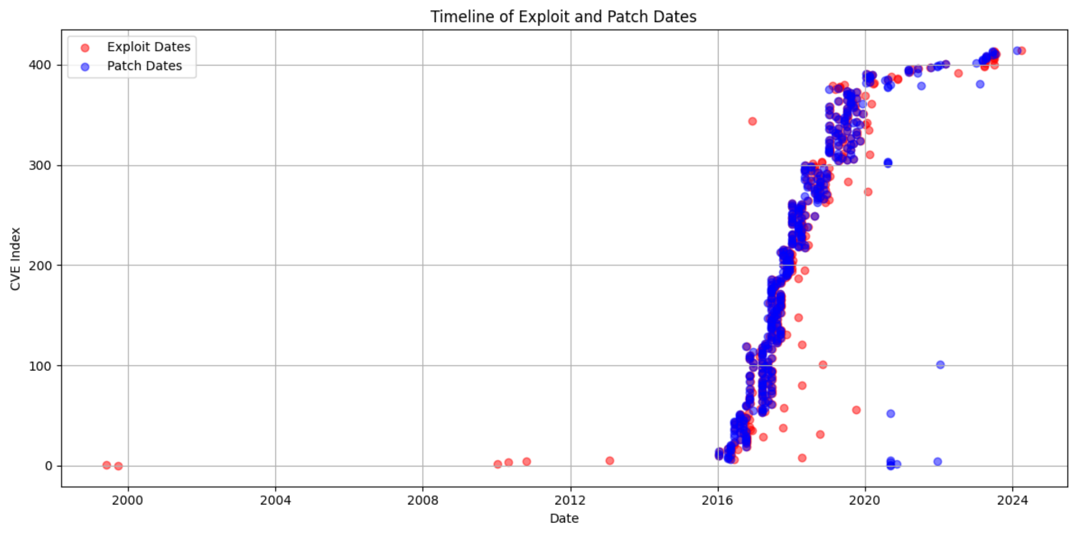


FIGURE 5.11: Gap between exploit and patch dates for Microsoft-related CVEs (2016-2024).

The analysis reveals that 26 CVEs (6.3%) were exploited before being patched, while 389 (93.7%) were patched before being exploited. This distribution suggests that despite challenges in the patching process, the majority of vulnerabilities in the Microsoft ecosystem had patches available before exploitation occurred. However, the small percentage of pre-patch exploits represents a critical security concern, as these cases indicate successful exploitation during the window of vulnerability.

The temporal patterns in Figure 5.11 reveal periods of heightened exploitation activity, particularly during 2017-2019, followed by more consistent patching performance in recent years. This visualization addresses a key limitation in existing literature by providing empirical evidence of the "race" between attackers and defenders in a major software ecosystem.

The complete lifecycle analysis extends Frei's foundational model [46] by providing detailed empirical data on the temporal relationships between key vulnerability events. While Frei's work established the conceptual framework for vulnerability lifecycles, our analysis quantifies these relationships in a contemporary security context, enabling more precise understanding of vulnerability management dynamics.

The analysis also reveals that the median time from CVE reservation to patch availability is 134 days across all severity levels, with substantial variations based on severity (as detailed in Section 5.3.2). This overall timeline challenges assertions by Shahzad et al. [134] regarding increasingly efficient patch development processes, suggesting that significant improvements are still needed in vulnerability remediation timelines.

Key Insights from Integrated Lifecycle Analysis

**1. Full Vulnerability Lifecycle Timeline:**

- The typical Microsoft vulnerability progresses from CVE reservation to exploit publication in 141 days (median), with patches typically appearing 7 days before exploits (median of Patch - Exploit = -7 days).
- Despite the negative median indicating most patches precede exploits chronologically, the significant range and skewness in the distributions highlight the unpredictable nature of vulnerability lifecycles.

**2. The "Race" Between Attackers and Defenders:**

- 93.7% of vulnerabilities were patched before being exploited, suggesting generally effective security response processes.
- The critical 6.3% of vulnerabilities exploited before patching represent significant security risks, highlighting the importance of rapid security response.

**3. Microsoft vs. Third-Party Software Patterns:**

- Microsoft-specific products show more predictable lifecycle patterns than third-party applications operating on Microsoft platforms.
- Third-party applications exhibit greater variability in patching timelines, with extreme outliers suggesting coordination challenges across vendor boundaries.

**4. Temporal Evolution:**

- Recent years (2020-2024) show improved consistency in patching relative to exploitation, suggesting maturation in vulnerability management processes.
- Historical periods of heightened exploitation (2017-2019) align with substantial increases in CVE volumes, indicating scaling challenges in vulnerability management.

### 5.3.2 CVE Lifecycle Event Differences by Severity

Understanding the differences in lifecycle events based on vulnerability severity provides critical insights for security prioritization and resource allocation. This section analyzes how severity levels influence the timing relationships between key lifecycle events.

#### 5.3.2.1 Severity-Based Lifecycle Timing Analysis

Table 5.1 presents the mean, median, and mode differences between three key lifecycle events-Exploit Added - CVE Creation, Patch Availability - Exploit Added, and Patch Availability - CVE Creation-categorized by severity level.

| Event | Statistic | Critical | High | Medium | Low |
|---|---|---|---|---|---|
| Exploit - Creation | Mean | 227.68 | 179.73 | 152.96 | 147.75 |
| | Median | 192.00 | 141.00 | 138.00 | 141.00 |
| | Mode | 192 | 226 | 106 | 141 |
| Patch - Exploit | Mean | 33.58 | 62.32 | 61.07 | -24.50 |
| | Median | -6.00 | -7.00 | -9.00 | -11.00 |
| | Mode | -6 | -7 | -9 | -70 |
| Patch - Creation | Mean | 261.26 | 242.05 | 214.03 | 123.25 |
| | Median | 186.00 | 134.00 | 129.00 | 130.00 |
| | Mode | 186 | 219 | 97 | 71 |

TABLE 5.1: Lifecycle Events Differences by Severity (Days)

The analysis reveals several counterintuitive patterns in vulnerability lifecycle events across severity levels:

- **Critical Severity:** Critical vulnerabilities show the longest delay between CVE creation and exploit publication, with a median of 192 days-substantially longer than all other severity levels. The median time from CVE creation to patch availability is also highest for critical vulnerabilities at 186 days. This contradicts conventional expectations that more severe vulnerabilities would be exploited more rapidly.

- **High Severity:** High-severity vulnerabilities show intermediate exploitation timelines (median 141 days) and patch development times (median 134 days from CVE creation). The gap between exploit publication and patch availability remains negative (median -7 days), indicating exploits typically precede patches.

- **Medium Severity:** Median exploitation time for medium-severity vulnerabilities is shortest at 138 days, suggesting these vulnerabilities are exploited relatively quickly despite their lower severity rating. The median time from CVE creation to patch availability is 129 days, slightly faster than for high-severity issues.

- **Low Severity:** Low-severity vulnerabilities show a median exploitation time of 141 days-equivalent to high-severity vulnerabilities and longer than medium-severity ones. However, they show the most negative median gap between patch and exploit (-11 days), indicating these vulnerabilities typically experience the longest window of vulnerability between exploit and patch.

The counterintuitive observation that critical-severity vulnerabilities exhibit longer exploitation timelines (median 192 days) than lower-severity issues challenges fundamental assumptions in vulnerability prioritization frameworks such as EPSS [68]. This finding extends research by Costa et al. [31], who identified limitations in current severity scoring systems for predicting actual exploitation patterns. The negative median differences between patch availability and exploit addition across all severity levels (-6 to -11 days) indicates that the "race" between attackers and defenders frequently favors attackers, regardless of severity level.

The overall time from CVE creation to patch availability shows a clear severity-based pattern, with critical vulnerabilities taking longest to patch (median 186 days) followed by high, medium, and low severity. This pattern suggests that patch development time may be influenced by vulnerability complexity rather than simply prioritization based on severity rating.

### 5.3.2.2 Temporal Trends in Lifecycle Events by Severity

To understand how vulnerability lifecycle events evolve over time for different severity levels, we analyzed yearly trends from 2016 to 2024. Figures 5.12 through 5.15 present these trends for each severity level.

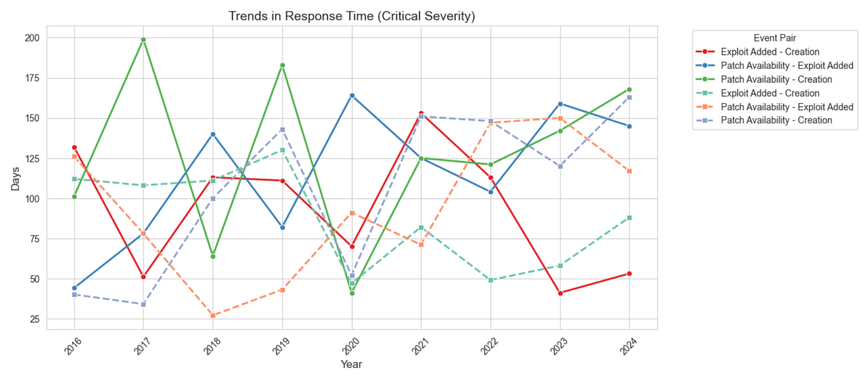


FIGURE 5.12: Yearly trend of mean and median time differences for *Critical* severity across all events. Dashed lines represent median values, and solid lines represent mean values.

Figure 5.12 shows the yearly trends for critical vulnerabilities. The time between CVE creation and exploit addition has generally increased over time, with a notable reduction in 2023. Patch release times have seen significant fluctuations, with shorter timelines in the most recent years (2021-2023), suggesting improvements in the response to critical vulnerabilities.

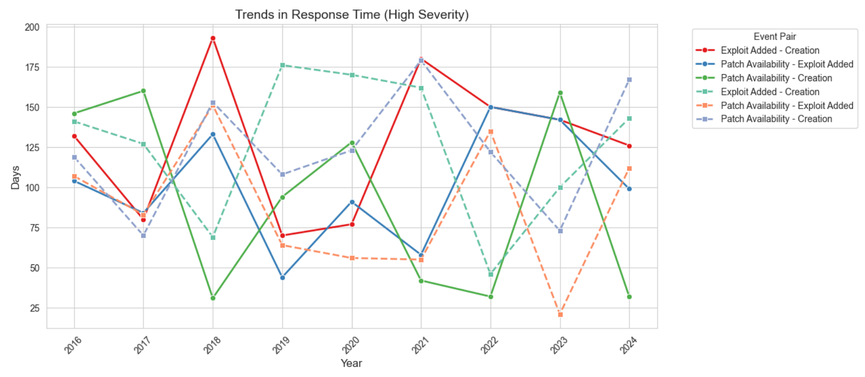


FIGURE 5.13: Yearly trend of mean and median time differences for *High* severity across all events. Dashed lines represent median values, and solid lines represent mean values.

Figure 5.13 illustrates trends for high-severity vulnerabilities. Similar to critical vulnerabilities, high-severity issues have faced delays in patching, though recent years (2021-2023) show some improvements in response to exploits, with patches becoming available sooner after exploit disclosure.

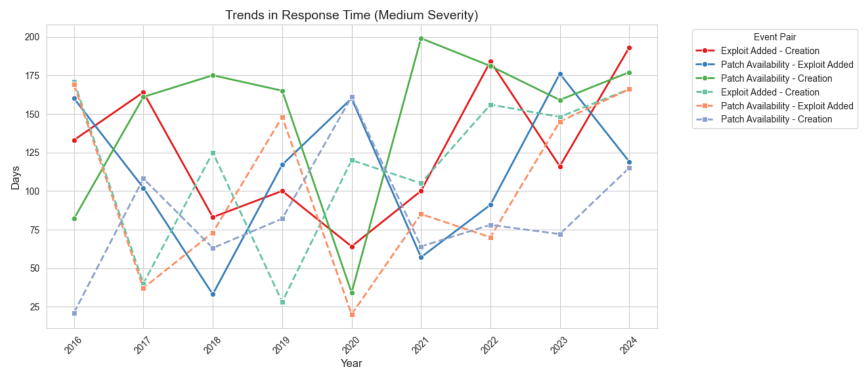


FIGURE 5.14: Yearly trend of mean and median time differences for *Medium* severity across all events. Dashed lines represent median values, and solid lines represent mean values.

Figure 5.14 depicts trends for medium-severity vulnerabilities. The data reveals a consistent issue with patch delays, with the time between CVE creation and patch availability remaining relatively high throughout the years, highlighting ongoing challenges in patching medium-severity vulnerabilities despite their shorter exploitation timelines.

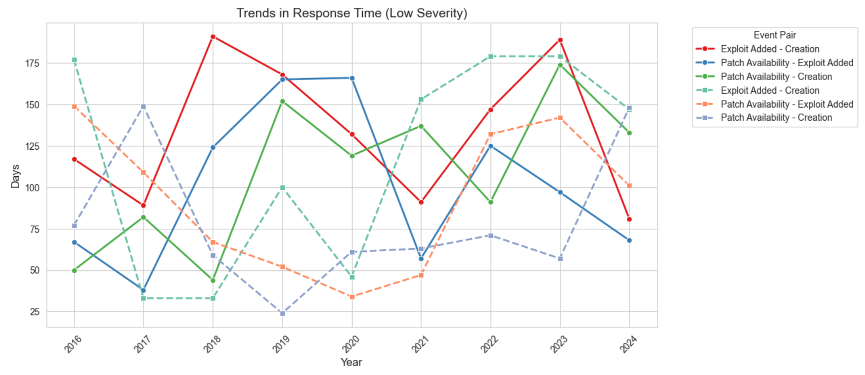


FIGURE 5.15: Yearly trend of mean and median time differences for *Low* severity across all events. Dashed lines represent median values, and solid lines represent mean values.

Figure 5.15 shows trends for low-severity vulnerabilities, which generally exhibit shorter timelines for patch availability, particularly between CVE creation and patch release. This pattern reflects increased efficiency in patching lower-risk vulnerabilities, though variability remains substantial.

The analysis of temporal trends reveals important insights into the evolution of vulnerability management practices:

- **Increasing delays in exploit timelines for critical vulnerabilities**: The gap between CVE creation and exploit availability has grown over time for critical vulnerabilities, suggesting increased complexity in exploitation or improved defenses against immediate exploitation.

- **Persistent patch delays**: Patches are frequently delayed relative to exploit publication, particularly for critical and high-severity vulnerabilities, highlighting ongoing challenges in timely remediation.

- **Improved patch timelines for low-severity vulnerabilities**: The time between CVE creation and patch release for low-severity vulnerabilities has shortened in recent years, suggesting more efficient processes for non-critical issues.
- **Fluctuations in patching efficiency**: The significant reduction in patching time for critical vulnerabilities in 2023 suggests potential improvements in vulnerability management processes or shifts in Microsoft's patching approach.

The fluctuations in patching efficiency for critical vulnerabilities, with significant improvements observed in 2023, align with research by Costa et al. [31] on the evolving nature of vulnerability management practices. The distinct temporal patterns across different severity levels support Roytman and Jacobs' [129] observations about the complex interplay between vulnerability characteristics and remediation timelines.

#### 5.3.2.3 Three-Way Comparison of Patching Times by Severity

To provide comprehensive context for Microsoft's patching patterns, we conducted a three-way comparison visualized in Figure 5.16. This analysis illustrates the differences between all Microsoft patched CVEs, non-exploited patched CVEs, and exploited patched CVEs across severity levels.

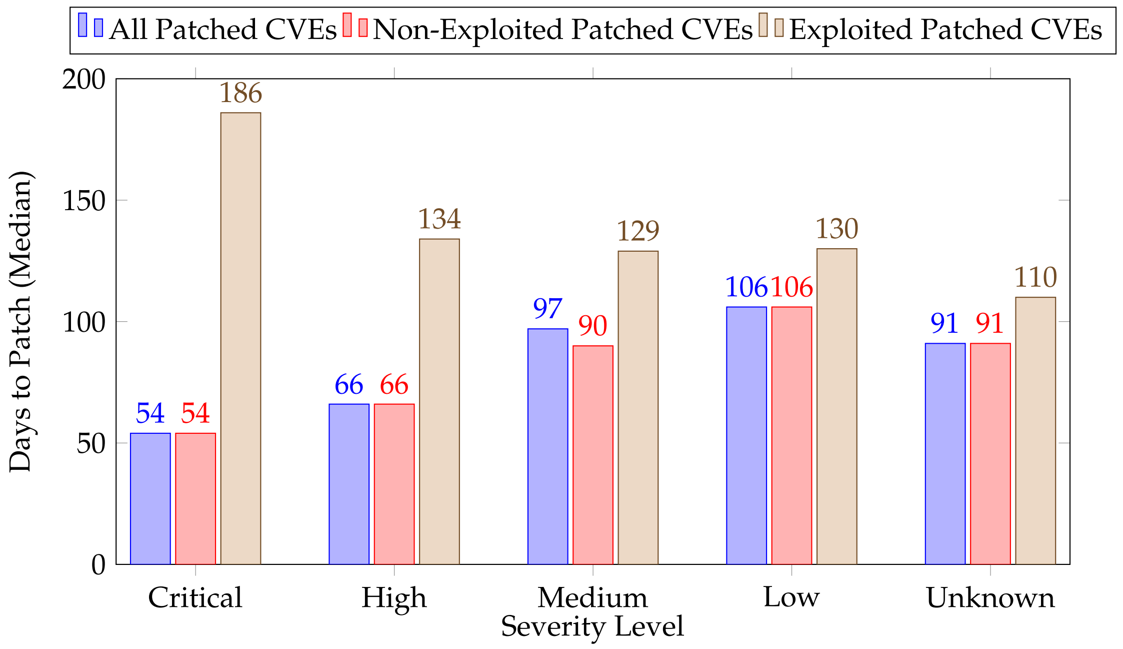


FIGURE 5.16: Three-Way Comparison: Median Patching Times (in days) for All Microsoft Patched CVEs, Non-Exploited Patched CVEs, and Exploited Patched CVEs by Severity Level

This comparison reveals striking differences between exploited and non-exploited vulnerabilities:

- **Critical Vulnerabilities**: Exploited critical vulnerabilities show a median patching time of 186 days, compared to just 54 days for non-exploited critical vulnerabilities-a 244% increase. This substantial difference suggests that exploited critical vulnerabilities may possess characteristics that make them particularly challenging to patch.

- **High Severity**: Exploited high-severity vulnerabilities require a median of 134 days to patch, compared to 66 days for non-exploited ones-a 103% increase.
- **Medium Severity**: Exploited medium-severity vulnerabilities are patched in a median of 129 days, compared to 90 days for non-exploited ones-a 43% increase.
- **Low Severity**: Exploited low-severity vulnerabilities require a median of 130 days to patch, compared to 106 days for non-exploited ones-a 23% increase.

This analysis reveals that the patching time gap between exploited and non-exploited vulnerabilities increases with severity level. For critical vulnerabilities, the gap is most pronounced, with exploited vulnerabilities taking more than three times longer to patch than non-exploited ones.

The minimal difference between "All Patched CVEs" and "Non-Exploited Patched CVEs" indicates that the overall patching timeline is largely driven by non-exploited vulnerabilities, which represent the vast majority of cases. The substantial deviation for exploited vulnerabilities suggests these cases follow fundamentally different lifecycle patterns, potentially due to their complexity, detection circumstances, or specific characteristics that make them targets for exploitation.

The significant discrepancy between median patching times for exploited critical vulnerabilities (186 days) versus all patched critical vulnerabilities (54 days) represents a 244% difference that aligns with Allodi's [8] research on heavy-tailed distributions in vulnerability exploitation. This substantial gap supports his assertion that exploited vulnerabilities often represent a distinct subset with unique characteristics that differ from the broader vulnerability population.

Key Insights from Severity-Based Lifecycle Analysis

**1. Counterintuitive Severity-Exploitation Relationship:**

- Critical vulnerabilities exhibit the *longest* delay between CVE creation and exploitation (median 192 days), contradicting assumptions that higher-severity vulnerabilities are exploited more rapidly.
- Medium-severity vulnerabilities show the shortest exploitation timeline (median 138 days), suggesting attackers may target "easier" vulnerabilities rather than the most severe ones.

**2. Exploitation Consistently Precedes Patching:**

- Negative median differences between patch availability and exploit addition across all severity levels (-6 to -11 days) indicate that attackers frequently gain an advantage in the "race" between exploitation and patching.
- Low-severity vulnerabilities show the longest window of vulnerability between exploit and patch (median -11 days), suggesting these issues may receive lower priority in security response.

**3. Dramatic Patching Disparities for Exploited vs. Non-Exploited CVEs:**

- Exploited critical vulnerabilities take 244% longer to patch than non-exploited ones (186 vs. 54 days), with the disparity decreasing as severity decreases.
- This pattern suggests that exploited vulnerabilities likely possess unique characteristics that complicate patch development.

**4. Temporal Improvements in Recent Years:**

- Significant reductions in patching time for critical vulnerabilities in 2023 suggest recent improvements in vulnerability management processes.
- The relative stability of exploitation timelines across severity levels in recent years indicates established patterns in how attackers identify and target vulnerabilities.

## 5.4 Special Cases and Outliers in Vulnerability Lifecycles

While central tendency measures provide valuable insights into typical vulnerability lifecycle patterns, the examination of outliers often reveals critical edge cases that highlight systemic challenges in vulnerability management. This section analyzes extreme cases in the CVE lifecycle, focusing particularly on vulnerabilities with exceptionally long timeframes between key events. Understanding these outliers provides important context for the limitations of current vulnerability management approaches and the challenges in coordinating security responses across complex software ecosystems.

### 5.4.1 Outlier Investigation in CVE Lifecycle Events Distribution

Analyzing outliers within the lifecycle distribution of vulnerabilities reveals significant deviations from typical patterns. These outliers, which exhibit extreme gaps between lifecycle events, can offer valuable insights into unusual delay patterns, such as those between discovery, patch issuance, and the first known exploit of a vulnerability.

In our dataset of 415 CVEs with exploits and patches, we identified several outliers with lifecycle differences exceeding 1,000 days. These cases represent the extreme tail of the distribution and merit detailed examination to understand their causes and implications.

#### 5.4.1.1 Lifecycle Outliers in Microsoft-Specific Products

Out of the 391 CVEs directly related to Microsoft products, only one CVE (CVE-2017-0148) exhibited a difference exceeding 1,000 days between exploitation and CVE creation date, as detailed in Table 5.2.

| CVE ID | Date Reserved | Exploit date added | Patch Release Date | CVSS Score | CWE | CVE Base Severity | Microsoft Severity | Affected Products |
|---|---|---|---|---|---|---|---|---|
| CVE-2017-0148 | 9/9/2016 | 10/2/2019 | 3/14/2017 7:00 | 8 | CWE-20 | Critical | High | ['windows_rt_8.1', 'server_message_block', 'windows_10', 'windows_8.1','windows_10', 'windows_server_2012', 'windows_7', 'windows_server_2008','windows_server_2012', 'windows_server_2016', 'windows_vista','windows_server_2008', 'windows_10'] |

TABLE 5.2: CVE with more than 1000 days between Exploitation and CVE Creation Date

This exceptional case involved a vulnerability in Microsoft's Server Message Block (SMB) protocol with a CVSS score of 8.0. The CVE was reserved in September 2016, patched in March 2017 (188 days later), but an exploit was only documented in ExploitDB in October 2019-more than three years after the CVE reservation. This unusual pattern could be attributed to several factors:

- The vulnerability might have been difficult to exploit initially, with exploitation techniques only emerging later.
- The exploit may have existed earlier but was not publicly documented until 2019.
- The long gap might reflect strategic decisions by attackers to delay public disclosure of exploitation techniques for a critical vulnerability.

This case represents an anomaly in Microsoft's vulnerability lifecycle patterns, as it deviates significantly from the typical exploitation timeline (median 141 days). The absence of other similar cases among Microsoft-specific products suggests generally consistent and predictable lifecycle patterns for Microsoft vulnerabilities.

#### 5.4.1.2 Lifecycle Outliers in Third-Party Applications

A more concerning pattern emerged in the analysis of the 24 CVEs related to third-party applications operating on Microsoft platforms. Nine of these CVEs (37.5%)

exhibited differences exceeding 1,000 days between patch availability and exploit publication dates, as detailed in Table 5.3.

| CVE ID | Date Updated | Date Reserved | Exploit date added | Patch ReleaseDate | CVSS Score | CWE | CVE Base Severity | Microsoft_Severity | Product |
|---|---|---|---|---|---|---|---|---|---|
| CVE-2016-6664 | 10/9/2018 18:57 | 8/10/2016 | 11/1/2016 | 9/8/2020 7:00 | 7 | CWE-59 | High | High | NA |
| CVE-2009-4487 | 10/10/2018 18:57 | 12/30/2009 | 1/11/2010 | 11/10/2020 8:00 | NA | NVD-CWE-noinfo | Medium | NA | NA |
| CVE-2013-0221 | 11/23/2013 18:28 | 12/6/2012 | 1/21/2013 | 9/8/2020 7:00 | NA | CWE-20 | Medium | NA | NA |
| CVE-1999-0236 | 8/17/2022 7:00 | 6/7/1999 | 9/25/1999 | 9/8/2020 7:00 | 8 | CWE-200 | High | High | NA |
| CVE-2016-7567 | 11/10/2018 10:57 | 9/9/2016 | 11/9/2018 | 1/11/2022 8:00 | 10 | CWE-119 | Critical | Critical | NA |
| CVE-2009-4484 | 1/4/2018 18:57 | 12/30/2009 | 4/30/2010 | 9/8/2020 7:00 | NA | CWE-787 | High | NA | NA |
| CVE-2010-2891 | 10/10/2018 18:57 | 7/27/2010 | 10/20/2010 | 12/14/2021 8:00 | NA | CWE-119 | High | NA | NA |
| CVE-2019-15126 | 8/11/2020 18:08 | 8/17/2019 | 3/19/2020 | 2/14/2023 8:00 | NA | CWE-367 | Low | NA | NA |
| CVE-1999-1412 | 3/22/2002 10:00 | 8/31/2001 | 6/3/1999 | 9/8/2020 7:00 | NA | NVD-CWE-noinfo | Medium | NA | NA |

TABLE 5.3: CVEs with more than 1000 days between Patching and Exploitation Date

These outliers share several notable characteristics:

- Most affected products for these CVEs are unspecified in the Microsoft data ("NA"), suggesting limited visibility into third-party components within Microsoft's ecosystem.
- The average CVSS score for these CVEs is 6.5, indicating moderate to high severity despite the extreme delay in patching.
- The most extreme case (CVE-1999-0236) involved a high-severity vulnerability with a CVSS score of 8.0, where the patch was released nearly 21 years after the exploit was published.
- Several of these vulnerabilities affect common third-party components like database systems (e.g., CVE-2016-6664 affecting MySQL/MariaDB) and web servers.

The high proportion of extreme outliers among third-party applications (37.5% compared to just 0.26% in Microsoft-specific products) highlights significant challenges in cross-vendor vulnerability management. This disparity raises two critical questions:

1. **Why do these vulnerabilities experience such extreme patching delays?** The delays likely result from coordination challenges between different vendors, limited visibility into dependencies, and potential gaps in responsibility for security remediation.
2. **Why do vulnerabilities in third-party applications appear in Microsoft's patching process?** Microsoft's responsibility extends to the security of their entire ecosystem, including dependencies on third-party components. As documented by Microsoft, these vulnerabilities were patched as part of their security updates despite originating in non-Microsoft products.

### 5.4.2 Analysis of Specific Outlier Cases

To better understand the factors contributing to extreme lifecycle outliers, we examined several representative cases in detail.

#### 5.4.2.1 Case Study: CVE-2016-6664 (MySQL/MariaDB)

CVE-2016-6664 represents a notable case where a high-severity vulnerability in a database system (MySQL/MariaDB) was actively exploited in November 2016, shortly after its disclosure in August 2016. While the original vendors (Oracle, MySQL, MariaDB, Percona) released patches in November 2016, Microsoft's patch for this vulnerability was only documented in September 2020-nearly four years later.

This substantial delay highlights several key challenges:

- **Dependencies in Complex Ecosystems**: Microsoft products may include or interact with third-party components like MySQL, creating complex dependency chains that complicate vulnerability management.
- **Visibility Limitations**: Organizations may have limited visibility into vulnerabilities affecting components they did not develop, particularly when these components are deeply integrated into their systems.
- **Coordination Challenges**: Cross-vendor coordination for vulnerability remediation introduces substantial complexity and potential delays.

This case exemplifies the "long tail" of vulnerability remediation described by Allodi [8], where certain vulnerabilities exhibit dramatically different lifecycle patterns compared to the broader population.

#### 5.4.2.2 Case Study: CVE-1999-0236 (Information Disclosure)

Perhaps the most extreme outlier in our dataset, CVE-1999-0236 involves a high-severity information disclosure vulnerability initially disclosed in June 1999, with an exploit published in September 1999. Microsoft's patch for this vulnerability was only documented in September 2020-almost 21 years after exploit publication.

This extraordinary case represents an unusual situation where a decades-old vulnerability remained unpatched in some part of Microsoft's ecosystem. Possible explanations include:

- **Legacy Component Integration**: The vulnerability may have affected a legacy system or component that was subsequently integrated into Microsoft's ecosystem.
- **Incomplete Prior Remediation**: The vulnerability might have been partially addressed in earlier patches, with complete remediation only documented in 2020.
- **SBOM Completeness**: The inclusion of this extremely old vulnerability in recent patch documentation may reflect Microsoft's efforts to maintain a comprehensive Software Bill of Materials (SBOM) across their ecosystem.

This case highlights the persistent challenges of managing security in complex, evolving software ecosystems with long-lived components and dependencies. It exemplifies the findings of Akhoundali et al. [4], who documented how vulnerabilities often require multiple fixes over extended periods, particularly when they span across multiple components or vendors.

### 5.4.3 Factors Contributing to Extreme Lifecycle Delays

Our analysis of outlier cases reveals several key factors that contribute to extreme delays in vulnerability lifecycles, particularly in the context of Microsoft's ecosystem:

#### 5.4.3.1 Complexity of Patch Development

Some vulnerabilities, especially those affecting core components of a system or software, require complex patches that take considerable time to develop and test. This complexity may be particularly pronounced for:

- Vulnerabilities affecting low-level system components where changes might impact a wide range of functionality
- Issues spanning multiple products or requiring coordinated fixes across different components
- Vulnerabilities where simple fixes might introduce compatibility issues or regressions

Ivanti's research [65] documented this challenge, finding that 71% of IT and security professionals reported patching to be "overly complex, cumbersome, and time-consuming." Their study highlighted how remote work environments have exacerbated these challenges, with 57% of respondents indicating that remote work has made patching more difficult. The complexity of modern IT environments-with diverse devices, operating systems, and applications-creates substantial challenges for comprehensive vulnerability remediation.

#### 5.4.3.2 Lack of Public Awareness or Pressure

In some cases, vulnerabilities may not be promptly addressed due to limited public awareness or perceived urgency. CVE-2019-15126, which showed a 172-day gap between reservation and publication, exemplifies how awareness and subsequent action on vulnerabilities can progress slowly, affecting their overall lifecycle.

The absence of public pressure or active exploitation may reduce the perceived urgency for patching, particularly for vulnerabilities in less visible components. This factor aligns with research by Brilhante et al. [27], who identified how public attention significantly influences vulnerability remediation timelines.

#### 5.4.3.3 Administrative and Process Delays

Administrative and process delays represent another significant factor in extended vulnerability lifecycles, particularly in complex organizational environments. Miller et al. [105] documented how organizational inefficiencies, including complex approval processes and inadequate resource allocation, can lead to missed patches and prolonged exposure to vulnerabilities. Their research specifically noted that "delays in the administrative processes affected the patch deployment," highlighting how internal challenges can exacerbate security risks.

These administrative delays can be particularly pronounced in cross-vendor scenarios, where coordination between different organizations introduces additional complexity and potential bottlenecks. As noted by Microsoft Tech Community [26], delays in vulnerability database processes have caused thousands of vulnerabilities to lack crucial metadata, further complicating effective patch management.

#### 5.4.3.4 Third-Party Component Management

The most dramatic outliers in our analysis involved third-party components integrated into Microsoft's ecosystem. This pattern highlights the particular challenges of managing vulnerabilities across vendor boundaries. These challenges include:

- **Responsibility Ambiguity**: Uncertainty regarding which party is responsible for patching vulnerabilities in integrated components
- **Dependency Awareness**: Limited visibility into the full dependency chain within complex software ecosystems
- **Integration Complexity**: Challenges in integrating third-party patches into existing systems without introducing new issues

Recent initiatives like the Software Bill of Materials (SBOM) aim to address these challenges by providing transparent documentation of all components in software products. However, our findings suggest that significant challenges remain in coordinating vulnerability management across organizational boundaries.

Key Insights from Outlier Analysis

**1. Dramatic Contrast Between Microsoft and Third-Party Outliers:**

- Only 0.26% of Microsoft-specific products (1 out of 391) exhibited extreme lifecycle delays exceeding 1,000 days.
- In stark contrast, 37.5% of third-party applications (9 out of 24) showed such extreme delays, highlighting significant challenges in cross-vendor vulnerability management.

**2. Extreme Patching Delays for Legacy Vulnerabilities:**

- The most extreme case (CVE-1999-0236) remained unpatched for nearly 21 years after exploit publication despite its high severity (CVSS 8.0).
- Several other vulnerabilities from the early 2000s were only patched in 2020-2023, suggesting persistent challenges with legacy system security.

**3. Multi-Vendor Coordination Challenges:**

- Vulnerabilities like CVE-2016-6664 (MySQL/MariaDB) highlight how components developed by one vendor but integrated into another's ecosystem create complex dependency chains.
- These coordination challenges can leave vulnerabilities unpatched for years despite being actively exploited and patched by the original vendor.

**4. Complex Administrative and Technical Factors:**

- Multiple interrelated factors contribute to extreme lifecycle delays, including patch complexity, administrative processes, limited visibility into dependencies, and resource constraints.
- These factors exemplify the "long tail" of vulnerability remediation described in vulnerability research, where a small subset of cases exhibit dramatically different lifecycle patterns.

The extreme cases in our dataset, where patch delays extended beyond 1000 days, exemplify the "long tail" of vulnerability remediation documented by Allodi [8], who observed a heavy-tailed distribution in vulnerability exploitation and management. The persistence of administrative and process delays in the vulnerability lifecycle, even for critical issues, supports Gandhi et al.'s [51] systematic review findings on the significant organizational barriers to effective vulnerability management. These observations extend beyond technical limitations to encompass the complex socio-technical challenges in security response, as theorized by Anderson [11] in his economic analysis of cybersecurity incentives and constraints.

The presence of significant outliers in our dataset, particularly vulnerabilities with lifecycle differences exceeding 1000 days, represents an empirical validation of Franca et al.'s [45] assertion that traditional vulnerability management approaches contain fundamental effectiveness gaps. These extreme cases challenge the assumptions of standard vulnerability lifecycle models, such as those proposed by Frei [46], which tend to emphasize more typical remediation timelines. The concentration of these outliers among third-party components within Microsoft's ecosystem supports Akhoundali et al.'s [4] findings regarding the complexities of managing security fixes in interconnected software ecosystems.

## 5.5 Implications and Discussion

The comprehensive analysis of vulnerability lifecycles presented in this chapter yields significant implications for security practices, patch management strategies, and vulnerability prioritization frameworks. This section discusses these implications in the context of existing literature and identifies areas where current approaches may need refinement based on our empirical findings.

### 5.5.1 Theoretical Implications for Vulnerability Lifecycle Models

Our analysis both supports and extends Frei's foundational vulnerability lifecycle model [46]. While Frei's framework remains valid in identifying the key phases of vulnerability evolution (discovery, disclosure, exploitation, and patching), our findings suggest several important refinements to this model.

#### 5.5.1.1 Evolution in Exploitation-Disclosure Relationships

The dramatic evolution in the chronological relationship between disclosure and exploitation-from predominantly pre-CVE exploits in early years (1999-2004) to predominantly post-CVE exploits in recent years (2015-2024)-represents a fundamental shift not fully captured in traditional lifecycle models. This transformation suggests that vulnerability disclosure practices have matured significantly, with coordinated disclosure becoming the norm rather than the exception.

This finding challenges the assumption that vulnerability lifecycles follow consistent temporal patterns over time. Instead, our data indicates that these patterns evolve in response to changes in industry practices, security awareness, and coordination mechanisms. The shift is particularly pronounced in Microsoft's ecosystem, where pre-CVE exploits decreased from 98.5% in 1999-2000 to nearly zero after 2015, suggesting vendor-specific factors significantly influence lifecycle patterns.

#### 5.5.1.2 Severity-Lifecycle Relationship Complexity

Our findings challenge simplistic assumptions about the relationship between vulnerability severity and lifecycle timing. Contrary to expectations that higher-severity vulnerabilities would be exploited more rapidly, we found that critical vulnerabilities exhibited longer median times to exploitation (192 days) than medium-severity ones (138 days). This counterintuitive relationship suggests that severity ratings may not accurately predict exploitation likelihood or timing.

This complexity extends to patching timelines as well. While non-exploited vulnerabilities showed expected severity-based prioritization patterns (critical vulnerabilities patched faster than less severe ones), the relationship inverted for exploited vulnerabilities. Exploited critical vulnerabilities showed the longest median time to patch (186 days), suggesting that technical complexity or other factors beyond severity ratings significantly influence remediation timelines.

These findings align with Costa et al.'s [31] identification of limitations in current severity scoring systems. The significant gap we observed between patching times for exploited versus non-exploited vulnerabilities (particularly for critical issues) suggests that vulnerability lifecycle models must account for exploitation status as a key factor influencing remediation processes.

#### 5.5.1.3 The Long-Tail Distribution in Vulnerability Management

Our outlier analysis provides empirical support for Allodi's heavy-tailed distribution theory of vulnerability exploitation [8]. The presence of extreme outliers-particularly among third-party components-exemplifies how a small subset of vulnerabilities can exhibit dramatically different lifecycle patterns compared to the broader population.

These findings suggest that vulnerability lifecycle models should explicitly incorporate this heavy-tailed distribution, rather than focusing primarily on central tendencies. The dramatic contrast between typical patterns and extreme outliers (some spanning decades between exploitation and patching) highlights the limitations of models that assume relatively uniform processes across all vulnerabilities.

### 5.5.2 Practical Implications for Security Practitioners

Beyond theoretical considerations, our findings have several practical implications for security practitioners responsible for vulnerability management in organizational contexts.

#### 5.5.2.1 Exploit-Patch Gap Management

The consistent finding that exploits typically precede patches (median gaps of -6 to -11 days across severity levels) highlights the critical importance of compensating controls during the window of vulnerability. Security practitioners should recognize that patching alone may be insufficient, particularly for actively exploited vulnerabilities where the "race" between attackers and defenders frequently favors attackers.

These negative median values across all severity levels underscore the need for:

- Deployment of temporary mitigations during the patching process
- Implementation of behavioral detection mechanisms to identify exploitation attempts
- Application of defense-in-depth strategies to reduce vulnerability impact
- Continuous monitoring for signs of compromise during patching windows

The existence of this exploit-patch gap supports the findings of Woo et al. [161] regarding persistent challenges in timely patch development. Our quantification of this gap across different severity levels provides more precise guidance for security practitioners in allocating defensive resources based on empirically observed window lengths.

#### 5.5.2.2 Severity-Based Prioritization Refinement

Our findings suggest that traditional severity-based prioritization strategies may need refinement to better reflect real-world exploitation and patching patterns. The counterintuitive relationship between severity and exploitation timing indicates that security practitioners should consider multiple factors beyond CVSS scores when prioritizing vulnerability remediation:

- Exploitation status should significantly influence prioritization, with exploited vulnerabilities receiving expedited attention regardless of severity ratings
- Medium-severity vulnerabilities merit close attention given their shorter median time to exploitation
- Critical vulnerabilities in complex systems may require longer remediation timelines despite their high priority, necessitating robust interim mitigations

These refinements align with the EPSS approach [68], which incorporates multiple factors beyond severity scores to predict exploitation likelihood. Our findings extend this work by demonstrating empirically how exploitation status influences patching timelines across different severity levels.

#### 5.5.2.3 Third-Party Component Management

Perhaps the most striking practical implication emerges from our outlier analysis, which revealed dramatic differences between Microsoft-specific products and third-party components integrated into Microsoft's ecosystem. The high proportion of extreme outliers among third-party applications (37.5% versus 0.26% for Microsoft-specific products) highlights the critical importance of comprehensive dependency management.

Security practitioners should:

- Maintain comprehensive software bills of materials (SBOMs) to track all components in their environments, including third-party dependencies
- Implement vendor diversity risk assessments to identify potential chokepoints in vulnerability remediation processes
- Establish explicit responsibility assignments for timely remediation of vulnerabilities in integrated components
- Consider compensating controls for vulnerabilities in third-party components where patching delays may be extended

These recommendations address the challenges documented by Akhoundali et al. [4] regarding the complexities of managing security fixes in interconnected software ecosystems. Our findings provide empirical evidence of the dramatic lifecycle differences between vendor-specific and third-party vulnerabilities, underscoring the importance of cross-vendor coordination in effective security management.

### 5.5.3 Comparative Analysis with Prior Research

Our findings both support and challenge various aspects of prior research in vulnerability lifecycle analysis. This section examines how our results compare with key works in the field.

#### 5.5.3.1 Coordinated Disclosure Evolution

The dramatic shift from predominantly pre-CVE exploits in early years to post-CVE exploits in recent years aligns with Pastor et al.'s [117] suggestion that improved information sharing would lead to more structured vulnerability management. Our findings provide empirical evidence of this evolution, demonstrating that coordinated disclosure practices have indeed transformed the vulnerability landscape over the past two decades.

However, the persistence of negative median differences between patch and exploit dates (indicating that exploitation typically precedes patching) challenges the effectiveness of current coordinated disclosure mechanisms. This finding suggests that while disclosure coordination has improved, the ideal scenario of patches preceding exploitation remains elusive in practice.

#### 5.5.3.2 Severity-Exploitation Relationship

Our finding that critical vulnerabilities exhibit longer exploitation timelines than medium-severity ones extends Costa et al.'s [31] work on the limitations of severity scoring systems. While Costa et al. identified theoretical limitations in predictive capabilities, our research provides empirical evidence of the counterintuitive relationship between severity and exploitation timing.

This finding also aligns with Ruohonen's [130] observation that critical vulnerabilities often face longer assessment periods. Our research suggests this extended timeline applies not only to assessment but to the entire lifecycle, potentially because critical vulnerabilities often involve more complex systems or require more sophisticated exploitation techniques.

#### 5.5.3.3 Patch Development and Deployment Challenges

The significant gap between patching times for exploited versus non-exploited vulnerabilities supports Johnson et al.'s [70] research on the complexities of patch deployment in enterprise environments. Our finding that exploited critical vulnerabilities take 244% longer to patch than non-exploited ones provides quantitative evidence of these challenges, particularly for high-impact vulnerabilities.

This pattern also validates Xiong et al.'s [162] observations regarding the complex relationship between vulnerability disclosure and patch development. Their work highlighted how various factors beyond severity influence patch development timelines-a complexity demonstrated empirically in our analysis of patching patterns across different severity levels and exploitation statuses.

#### 5.5.3.4 Heavy-Tailed Distribution in Vulnerability Management

Our outlier analysis provides strong empirical support for Allodi's [8] heavy-tailed distribution theory. The concentration of extreme outliers among third-party components, with some patches delayed by more than a decade after exploitation, exemplifies the "long tail" of vulnerability remediation that Allodi described theoretically.

These findings extend Allodi's work by demonstrating that the heavy-tailed distribution applies not only to exploitation likelihood but to the entire vulnerability lifecycle, including remediation timelines. Furthermore, our research suggests this distribution is not uniform across all vulnerabilities but exhibits distinct patterns based on factors like component origin (vendor-specific versus third-party) and exploitation status.

### 5.5.4 Research Limitations and Future Directions

While our analysis provides valuable insights into vulnerability lifecycle patterns, several limitations should be acknowledged:

- **Exploit Data Completeness**: Our exploitation analysis relies on publicly documented exploits in the Exploit Database. This dataset may not capture private or undisclosed exploits, potentially understating the true extent of exploitation.
- **Vendor-Specific Focus**: The patch analysis focuses primarily on Microsoft-related vulnerabilities, which may exhibit different patterns than those affecting other vendors. Future research should extend this analysis to other major vendors to enable comparative assessment.
- **Temporal Coverage**: The detailed patch analysis covers 2016-2024, providing comprehensive coverage of recent vulnerability patterns but limiting historical comparisons for patching behaviors.
- **Third-Party Component Identification**: The limited product information for some third-party components constrains detailed analysis of specific applications and their exploitation patterns.

These limitations suggest several promising directions for future research:

- **Multi-Vendor Comparative Analysis**: Extending the three-point lifecycle analysis to other major vendors would provide valuable comparative insights into how different organizational practices influence vulnerability lifecycles.
- **Private Exploit Integration**: Incorporating data from private exploit datasets or vulnerability intelligence services could provide a more comprehensive view of exploitation patterns.
- **SBOM-Based Analysis**: As software bills of materials become more standardized and widely available, future research could leverage this information to analyze vulnerability patterns across different dependency types and integration models.
- **Exploitation Technique Evolution**: Analyzing how exploitation techniques have evolved over time could provide deeper insights into the relationship between vulnerability complexity and exploitation timing.

These future directions would build upon the foundation established in this chapter, further enhancing our understanding of vulnerability lifecycle dynamics and their implications for security practices.

## 5.6 Summary

This chapter has presented a comprehensive analysis of vulnerability lifecycles, examining the temporal relationships between key events including CVE reservation, exploit publication, and patch release. By analyzing these relationships across different dimensions-exploitation patterns, patching timelines, severity levels, and outlier cases-we have developed a more nuanced understanding of vulnerability management dynamics in contemporary security landscapes.

### 5.6.1 Key Findings

The analysis has yielded several significant findings that extend existing knowledge in vulnerability lifecycle research:

**Evolution in Vulnerability Disclosure and Exploitation Patterns**: The examination of CVE-exploit timing relationships from 1999 to 2024 revealed a dramatic shift from predominantly pre-CVE exploits in early years (1999-2004, over 80%) to post-CVE exploits in recent years (2015-2024, over 80%). This transformation reflects the maturation of coordinated vulnerability disclosure practices and improved information sharing across the security community. Microsoft's vulnerability landscape showed an even more pronounced evolution, shifting from 98.5% pre-CVE exploits in 1999-2000 to almost exclusively post-CVE exploits after 2015, demonstrating the effectiveness of structured security response processes.

**Exploitation-Patching Relationship**: Our analysis of 415 Microsoft-related vulnerabilities with both exploit and patch documentation revealed that exploits are typically published approximately 168 days after CVE creation (mean value, with a median of 141 days). The relationship between exploitation and patching showed a median gap of -7 days, indicating that exploits typically precede patches by about a week. This persistent negative gap highlights the ongoing challenge of developing and deploying timely patches, even in well-structured security environments. Nonetheless, 93.7% of vulnerabilities in our dataset were patched before being exploited, suggesting generally effective vulnerability management despite these challenges.

**Severity-Based Lifecycle Patterns**: Contrary to conventional expectations, critical vulnerabilities exhibited longer exploitation timelines (median 192 days) than medium-severity ones (median 138 days). This counterintuitive finding suggests that vulnerability complexity, rather than severity rating alone, may significantly influence exploitation patterns. Even more striking was the disparity in patching times between exploited and non-exploited vulnerabilities. Exploited critical vulnerabilities required a median of 186 days to patch, compared to just 54 days for non-exploited ones-a 244% increase. This substantial gap decreased with severity level, suggesting that exploited vulnerabilities, particularly at higher severity levels, may possess characteristics that significantly complicate remediation.

**Outliers and the Heavy-Tailed Distribution**: Our analysis of extreme outliers revealed dramatic differences between Microsoft-specific products and third-party components. Only 0.26% of Microsoft products (1 out of 391) exhibited extreme lifecycle delays exceeding 1,000 days, compared to 37.5% of third-party applications (9 out of 24). The most extreme case involved a high-severity vulnerability that remained unpatched for nearly 21 years after exploit publication. These findings provide empirical support for Allodi's [8] heavy-tailed distribution theory, demonstrating that vulnerability lifecycles do not follow uniform patterns but include significant outliers that require specialized management approaches.

**Product and Weakness Distribution Patterns**: Microsoft Office emerged as the product family with the highest number of vulnerabilities (6,342 patched CVEs), significantly ahead of Windows (4,869) and Microsoft .NET (4,867). This finding challenges common assumptions that operating systems represent the most vulnerable components in technology stacks. Analysis of Common Weakness Enumerations (CWEs) revealed that memory safety issues dominate Microsoft's vulnerability landscape, with CWE-787 (Out-of-bounds Write), CWE-416 (Use After Free), CWE-119 (Memory Buffer Restriction), and CWE-125 (Out-of-bounds Read) collectively accounting for 58.1% of Microsoft's top patched vulnerabilities-significantly higher than their 32.1% share in the broader vulnerability ecosystem.

**Temporal Evolution in Patching Efficiency**: Microsoft's patching activity increased dramatically from 386 patched CVEs in 2016 to a peak of 2,126 in 2022 (a 451% increase), before declining to 1,451 in 2023. This evolution reflects both the growing challenges of vulnerability management described by Johnson et al. [70] and the maturation of Microsoft's security response capabilities. Recent years show improved consistency in patching patterns and reduced temporal gaps between exploitation and remediation, suggesting enhancement in vulnerability management processes despite the increasing scale and complexity of the security landscape.

### 5.6.2 Theoretical and Practical Implications

The findings from this research extend vulnerability lifecycle theory in several important ways. While Frei's [46] foundational model remains valid in identifying key lifecycle phases, our research demonstrates that these patterns evolve significantly over time in response to changing industry practices. The shift from pre-CVE to post-CVE exploitation represents a fundamental transformation not fully captured in traditional static lifecycle models.

The counterintuitive relationship between vulnerability severity and lifecycle timing challenges simplistic risk assessment frameworks. Our findings support Costa et al.'s [31] identification of limitations in current severity scoring systems and suggest that more nuanced approaches to prioritization are needed-approaches that consider exploitation status alongside traditional severity metrics.

For security practitioners, our findings highlight several critical considerations:

- The consistent negative gap between exploit publication and patch availability (-6 to -11 days across severity levels) necessitates compensating controls during patching windows.
- Traditional severity-based prioritization should be refined to account for the empirically observed lifecycle patterns, with particular attention to medium-severity vulnerabilities given their shorter exploitation timelines.
- Third-party component management requires special attention, given the dramatically higher proportion of extreme outliers among these components compared to vendor-specific products.

### 5.6.3 Future Research Directions

While this research provides valuable insights into vulnerability lifecycle patterns, several promising directions for future research emerge:

- Extending the three-point lifecycle analysis to other major vendors would enable comparative assessment of how different organizational practices influence vulnerability management outcomes.
- Incorporating data from private exploit datasets or vulnerability intelligence services could provide a more comprehensive view of exploitation patterns beyond publicly documented exploits.
- Leveraging emerging standards for software bills of materials (SBOMs) could enable more detailed analysis of vulnerability patterns across different dependency types and integration models.

- Analyzing how exploitation techniques have evolved over time could provide deeper insights into the relationship between vulnerability complexity and exploitation timing.

These future directions would build upon the foundation established in this chapter, further enhancing our understanding of vulnerability lifecycle dynamics and their implications for security practices.

In conclusion, this chapter has provided a comprehensive empirical analysis of vulnerability lifecycles, quantifying the temporal relationships between key events and identifying significant patterns across dimensions of severity, product family, and vulnerability type. The findings extend existing theoretical frameworks while providing practical insights for improving vulnerability management practices in complex software ecosystems.

# CHAPTER 6

# FUTURE WORK

Building upon our comprehensive methodology (Chapter 3) and findings from our exploratory data analysis (Chapter 4) and lifecycle analysis (Chapter 5), this chapter outlines key directions for future research in vulnerability lifecycle analysis and management. Our findings have revealed several critical areas where additional investigation could significantly advance both theoretical understanding and practical applications in cybersecurity, with a particular focus on the temporal dynamics between vulnerability discovery, exploitation, and patching.

## 6.1 Expanding Data Sources

Our current analysis leverages data from Common Vulnerabilities and Exposures (CVE V5), ExploitDB, and Microsoft's Security Response Center (MSRC). While these sources have provided valuable insights, the dynamic nature of cybersecurity threats demands a more comprehensive data collection strategy. As demonstrated in Chapters 4 and 5, the integration of multiple data sources can provide richer insights into vulnerability lifecycles, particularly regarding the temporal relationships between discovery, exploitation, and patching.

### 6.1.1 Integration of MoreFixes Dataset

The recently published MoreFixes dataset [4] represents a significant advancement in vulnerability research that could substantially enhance our existing work. This dataset contains 26,617 unique CVEs with associated fix commits from 6,945 GitHub projects, representing a 397% increase in CVEs and a 295% increase in covered open-source projects compared to previous datasets. The MoreFixes dataset is particularly relevant to our research as it focuses on the relationship between CVEs and their fix commits, providing valuable temporal data about patch availability.

The MoreFixes dataset offers several advantages that address limitations in our current approach:

- Access to detailed information about security vulnerabilities in open-source projects, including precise fix commits. This could enhance our understanding of vulnerability remediation effectiveness, particularly for widely-used open-source components that may interact with Microsoft products.

- Comprehensive coverage across multiple programming languages, with 54 different file types related to programming languages and environment configurations. This would allow us to extend our analysis beyond our current focus on Microsoft-specific vulnerabilities.

- Rich metadata about repositories, commits, and changed methods that could provide deeper insights into the patching process. This is particularly relevant to our analysis of patching timelines in Section 5.3.1.2.

Based on our findings in Section 4.3.6, where we identified significant differences between vendor-reported exploits and independent databases, integrating the MoreFixes dataset could help bridge these gaps and provide a more complete picture of the vulnerability landscape, particularly regarding patch availability.

### 6.1.2 Integration of GitHub Advisory Database

The GitHub Security Advisory Database represents another valuable resource for vulnerability research that could complement our existing datasets. Our analysis in Chapter 4 revealed significant discrepancies between Microsoft-reported exploited CVEs and those listed in ExploitDB, highlighting the need for additional sources to create a more comprehensive view of the exploitation landscape.

The GitHub Advisory Database offers several advantages that address limitations in our current approach:

- Community-driven vulnerability reports that often precede official CVE entries, enabling earlier detection and analysis of emerging threats. This is particularly relevant for our temporal analysis, as it could provide more accurate timestamps for the discovery phase of vulnerabilities.
- Detailed contextual information through discussions and pull requests related to vulnerabilities, which could provide insights into the discovery and remediation process that are not available in traditional CVE descriptions.
- According to the MoreFixes study [4], the GitHub Security Advisory (GHSA) has proven to be particularly effective for gathering direct commit fixes and CVE to repository mappings.

### 6.1.3 Integration of Open Source Vulnerability Database (OSV)

The Open Source Vulnerability Database (OSV.dev) [54] represents another potential data source that could significantly enhance our analysis. OSV.dev focuses specifically on open-source vulnerabilities and provides structured data that could complement our existing sources. Recent observations from our analysis of outlier cases in Section 5.4.1 highlighted the need for better data on vulnerabilities in third-party components that affect Microsoft products.

Key advantages of OSV.dev include:

- Comprehensive coverage of vulnerabilities in open-source ecosystems, including those that might not have CVE identifiers.
- Precise version ranges for affected components, enabling more accurate tracking of vulnerability windows.
- Machine-readable data formats that facilitate automation and integration with other data sources.

### 6.1.4 Data Integration and Quality Challenges

While expanding our data sources offers significant potential benefits, our experiences with the current datasets highlight several challenges that must be addressed:

- Developing robust matching algorithms to correlate entries across different databases, particularly when dealing with vulnerabilities that may have different identifiers or descriptions in each source.
- Establishing data quality assessment and validation mechanisms to ensure the reliability of integrated information, building on our quality assurance methodology described in Section 3.6
- Creating standardized temporal representations to enable accurate lifecycle analysis across different sources, addressing the challenges identified in our timeline analysis in Section 5.3.2

The MoreFixes approach addresses many of these challenges through a novel workflow with several heuristic methods to enhance data quality, including a blocklist to filter out irrelevant repositories and a scoring system to evaluate the relevance of potential fix commits [4]. We will adopt similar techniques in our data integration approach to ensure high-quality data while expanding our sources.

## 6.2 Enhanced Lifecycle Analysis Approaches

Building upon our findings in Chapter 5, we have identified several opportunities to enhance our understanding of vulnerability lifecycles through more sophisticated analytical approaches. Our current analysis has revealed significant variations in the timing and sequence of key lifecycle events, but additional techniques could provide deeper insights into these patterns.

### 6.2.1 CVE-Exploit-Patch Race Analysis Framework

Our analysis in Section 5.3.1.3 identified a critical "race" between CVE reservation, exploit development, and patch deployment, with 6.3% of CVEs exploited before being patched. This finding highlights the importance of understanding the temporal dynamics of this three-way race. We propose developing a dedicated framework to analyze and predict the outcomes of these races, focusing on:

- Extending our current binary analysis (exploit before or after patch) to a more nuanced multi-state model that captures the full sequence of events (CVE reservation, exploit availability, patch release)
- Quantifying the time gaps between these key events and analyzing how these gaps have evolved over time, building on our initial findings in chapter 5.
- Identifying factors that influence the sequence of these events, particularly for the 6.3% of cases where exploits appeared before patches

This framework would directly extend our methodology from Section 5.3.1.3 to provide more actionable insights for vulnerability management. By integrating the MoreFixes dataset [4], we would gain access to a much larger corpus of patch data across multiple platforms, enabling more robust statistical analysis of these race conditions.

### 6.2.2 Time-to-Patch (TTP) Analysis Model

Our findings in Section 5.3.1.2 revealed significant variations in the time between exploit publication and patch availability, with a mean difference of approximately -31 days for Microsoft products. Building on this observation and leveraging the extensive patch data available in the MoreFixes dataset, we propose developing a dedicated model to analyze Time-to-Patch (TTP) metrics, focusing on:

- Comparing Microsoft patch deployment timelines with those of other major vendors represented in the MoreFixes dataset, which includes patch data for 6,945 unique GitHub projects
- Analyzing the relationship between vulnerability severity, exploitation status, and patching priority across different platforms and vendors
- Identifying specific factors that contribute to faster or slower patch development, such as vulnerability type (CWE), affected component, or programming language

This model would build directly on our findings in Section 5.3.2, which showed significant variations in patching timelines based on severity levels. The MoreFixes dataset, with its rich metadata about repositories, commits, and changed methods, would enable a much more detailed analysis of the patching process than is possible with our current data.

### 6.2.3 Heavy-Tailed Distribution Analysis

Our analysis in Chapter 4 revealed patterns in exploitation that align with research by Allodi [8], who demonstrated that vulnerability exploitation follows a heavy-tailed distribution where a small subset of vulnerabilities accounts for the majority of observed attacks in the wild. This pattern was particularly evident in our findings in Section 4.2.6, where we identified certain CVEs with disproportionately high numbers of exploits. Building on this insight, we propose:

- Applying heavy-tailed distribution models to our dataset to identify the characteristics of highly-exploited vulnerabilities.
- Analyzing the relationship between CWE types and exploitation rates, extending our analysis from Section 4.1.5.3 where we identified the most common CWEs.
- Examining whether the heavy-tailed distribution pattern holds across different platforms and vendors by comparing our Microsoft-focused findings with the broader data available in our dataset

This analysis would provide valuable insights for prioritizing vulnerability management efforts by identifying which vulnerabilities are most likely to be heavily exploited. By combining our exploitation data with the comprehensive patch data from Microsoft and MoreFixes, we could also analyze whether highly-exploited vulnerabilities receive faster patches, addressing a key question about vendor response priorities.

## 6.3 Predictive Modeling with Machine Learning

Based on the patterns identified in Chapters 4 and 5, we propose developing two targeted machine learning models to address specific aspects of vulnerability management. These models will directly build on our findings and leverage the expanded datasets described in Section 6.1.

### 6.3.1 Exploit Prediction Model

Our analysis in Section 5.2.1.3 revealed significant patterns in exploit timing relative to CVE reservations, with a clear shift toward post-CVE exploits in recent years (as shown in Figure 5.2). Building on these findings, we propose developing a machine learning model to predict exploit likelihood and timing for newly disclosed vulnerabilities.

#### 6.3.1.1 Model Design and Implementation

The proposed model would be designed as follows:

- **Feature Engineering**: We would extract features from CVE descriptions, CWE information, affected products, and severity scores. Based on our findings in Section 4.1.5.3, features related to CWE type would be particularly valuable, as certain weakness types (like CWE-79 and CWE-119) are more frequently exploited than others.

- **Temporal Features**: Our lifecycle analysis in Section 5.3.1.1 showed a mean difference of 168 days between CVE reservation and exploit publication, but with significant variation. We would incorporate temporal features such as time since disclosure, vendor response time, and historical exploitation patterns for similar vulnerabilities.

- **Training Data**: The model would be trained on our existing dataset of 468 Microsoft-related CVEs with exploits (identified in Section 4.3.6), supplemented with the broader exploitation data available in the MoreFixes dataset, which includes information on 26,617 unique CVEs.

#### 6.3.1.2 Technical Implementation

The model would use a gradient boosting approach (XGBoost) with the following implementation details outlined in Figure 6.1:

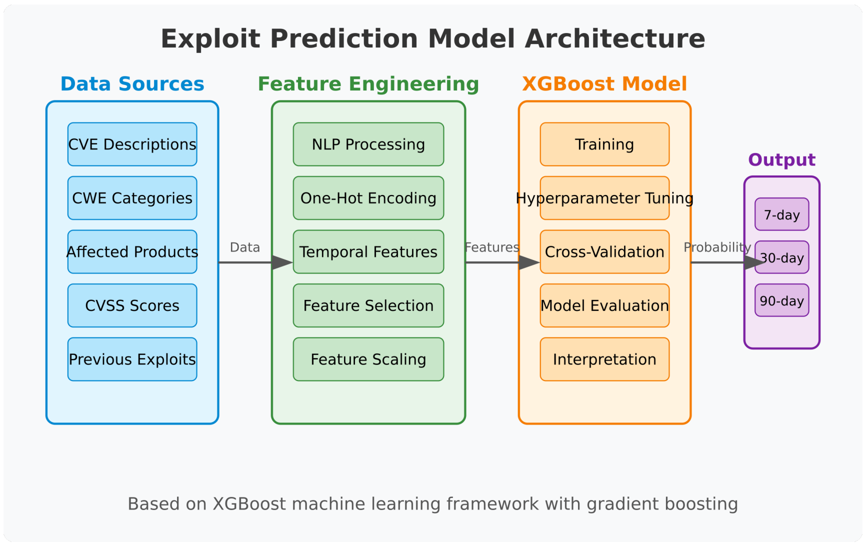


FIGURE 6.1: Illustration of the exploitation prediction model.

- **Input Features**:
  - Textual features extracted from CVE descriptions using NLP techniques
  - CWE category (encoded as one-hot vectors)
  - CVSS metrics (base, temporal, and environmental scores)
  - Affected product information
  - Vendor-specific features (e.g., Microsoft product type)
  - Temporal features (days since disclosure, patch availability status)
- **Output**: Probability of exploitation within specific timeframes (7 days, 30 days, 90 days) after disclosure
- **Evaluation Metrics**: Area Under the Receiver Operating Characteristic curve (AUROC), precision, recall, and F1-score with a focus on recall for high-severity vulnerabilities.
- **Model Selection Rationale**: We selected XGBoost as our primary algorithm due to its demonstrated superiority in vulnerability prediction tasks. Recent studies have shown that XGBoost achieves up to 94% accuracy in distinguishing between exploitable and non-exploitable vulnerabilities [18]. Its ensemble learning approach is particularly effective for handling the high-dimensional, non-linear relationships present in vulnerability data. Furthermore, XGBoost provides valuable feature importance metrics, allowing us to identify the most significant indicators of exploitation risk [22]. The algorithm's efficiency in processing large datasets makes it ideal for the continuous monitoring and updating necessary in dynamic vulnerability landscapes.

This model directly addresses the critical need "Race competency" identified in Section 5.3.1.3 to predict which vulnerabilities are likely to be exploited rapidly, enabling more effective prioritization of security responses, full illustration can be found on Appendix F.

### 6.3.2 Patch Prioritization Model

Our analysis in Section 5.3.2 revealed significant variations in patch release timelines based on vulnerability severity, with a median difference of -6 days between patch availability and exploit addition for critical vulnerabilities. While we don't have data on actual deployment dates within organizations (as these vary widely and are not publicly available), we can develop a model to optimize patch prioritization decisions based on the available patch release dates and exploit information.

#### 6.3.2.1 Model Design and Implementation

The proposed model would be designed as follows:

- **Feature Engineering**: We would extract features related to vulnerability characteristics, exploit status, and potential impact. Based on our findings in Section 5.3.2, features related to severity level would be particularly important, as different severity levels show distinct patterns in patch timing. According to research by Bozorgi et al. [25], the integration of exploit metadata can significantly improve vulnerability risk assessment.

- **Risk Modeling**: The model would incorporate risk assessment based on the heavy-tailed distribution of vulnerability exploitation identified in our analysis (Section 4.2.6) and supported by Allodi's research [8]. This approach aligns with the findings of Roytman and Jacobs [129], who demonstrated the importance of focusing resources on the small subset of vulnerabilities that pose the greatest risk.

- **Training Data**: The model would be trained on our existing dataset of Microsoft patch release dates, supplemented with the comprehensive patch data available in the MoreFixes dataset [4], which includes 31,883 unique fix commits. This dataset would provide a much broader view of patching patterns across different vendors and platforms.

#### 6.3.2.2 Technical Implementation

The model would use a supervised learning approach with the following implementation details, outlined in Figure 6.2:

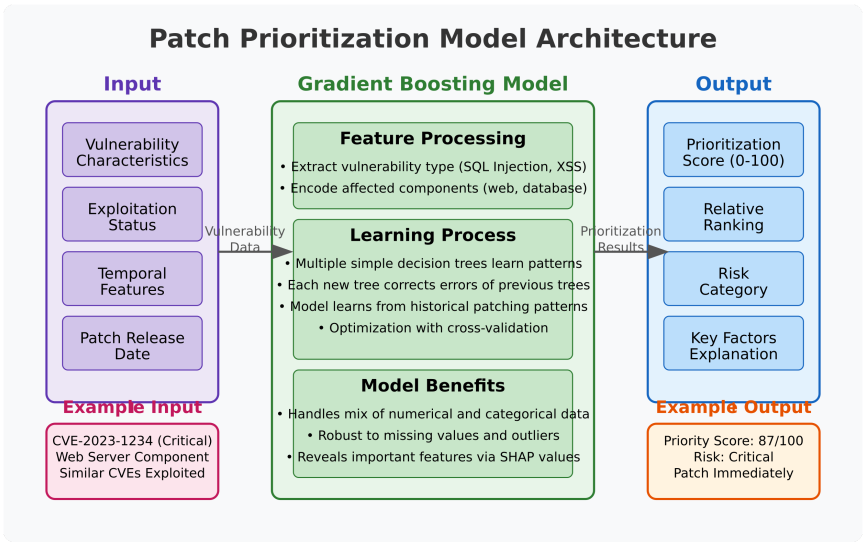

Patch Prioritization Model Architecture
Input
Vulnerability Characteristics
Exploitation Status
Temporal Features
Patch Release Date
Example Input
CVE-2023-1234 (Critical)
Web Server Component
Similar CVEs Exploited
Vulnerability Data
Gradient Boosting Model
Feature Processing
• Extract vulnerability type (SQL Injection, XSS)
• Encode affected components (web, database)
Learning Process
• Multiple simple decision trees learn patterns
• Each new tree corrects errors of previous trees
• Model learns from historical patching patterns
• Optimization with cross-validation
Model Benefits
• Handles mix of numerical and categorical data
• Robust to missing values and outliers
• Reveals important features via SHAP values
Prioritization Results
Output
Prioritization Score (0-100)
Relative Ranking
Risk Category
Key Factors Explanation
Example Output
Priority Score: 87/100
Risk: Critical
Patch Immediately

FIGURE 6.2: Illustration of the patch prioritization model.

- **Input Data (Features)**:
  - Vulnerability characteristics: The model will analyze information like the type of vulnerability (e.g., SQL injection, cross-site scripting), affected components, and severity scores.
  - Exploitation status: Whether the vulnerability is already being exploited, similar to ones that have been exploited, or shows characteristics that make it likely to be exploited soon.
  - Temporal features: How long the vulnerability has been known, how quickly similar vulnerabilities have been patched in the past.
- **Training Process**:
  - The model will learn from historical data how different factors relate to the importance of patching. For example, it might learn that certain types of vulnerabilities in internet-facing components need to be patched more urgently than similar vulnerabilities in internal components.
  - We'll use a technique called "gradient boosting" (specifically XGBoost), which builds multiple simple models that work together to make accurate predictions. This approach has been shown to work well for this type of problem in research by Jacobs et al. [68].
- **Output**: The model will produce a prioritization score for each vulnerability, indicating its relative importance for patching based on risk and impact.
- **Example Scenario**: For instance, the model would analyze a situation like:
  - Current unpatched vulnerabilities: CVE-2023-1234 (critical severity, web server), CVE-2023-5678 (high severity, database), CVE-2023-9012 (medium severity, workstation)
  - Available information: CVE-2023-1234 affects many systems, has no known exploits but similar vulnerabilities were exploited quickly; CVE-2023-5678 already has an exploit available but affects fewer systems; CVE-2023-9012 has been known for 60 days with no exploits

  - Model recommendation: A priority ranking of these vulnerabilities based on their risk characteristics

- **Algorithm Selection**: We specifically chose XGBoost for this task based on its proven performance in related cybersecurity prediction problems. XGBoost's ensemble learning capabilities allow it to effectively model the complex interactions between vulnerability characteristics and exploitation likelihood [128]. Additionally, the algorithm's robustness to noisy data is particularly valuable when working with vulnerability datasets, which often contain inconsistencies and missing information. Through hyperparameter optimization using techniques like Optuna [59], we can further enhance the model's accuracy and generalizability. This approach has demonstrated superior performance compared to traditional machine learning methods in similar prediction tasks [168].

This approach builds directly on the empirical findings from our lifecycle analysis. By providing data-driven recommendations for patch prioritization, the model would help organizations address the challenge identified in Section 5.3.1.2, where we observed that patches are often delayed even for critical vulnerabilities, full illustration can be found in Appendix F

## 6.4 Timeline and Research Plan

The implementation of our research agenda follows a structured approach over a 24-month period as appears in Figure 6.3, focusing on achievable goals within the scope of a doctoral thesis project. The timeline is designed to build incrementally on our existing work, with each phase addressing specific aspects of the research questions.

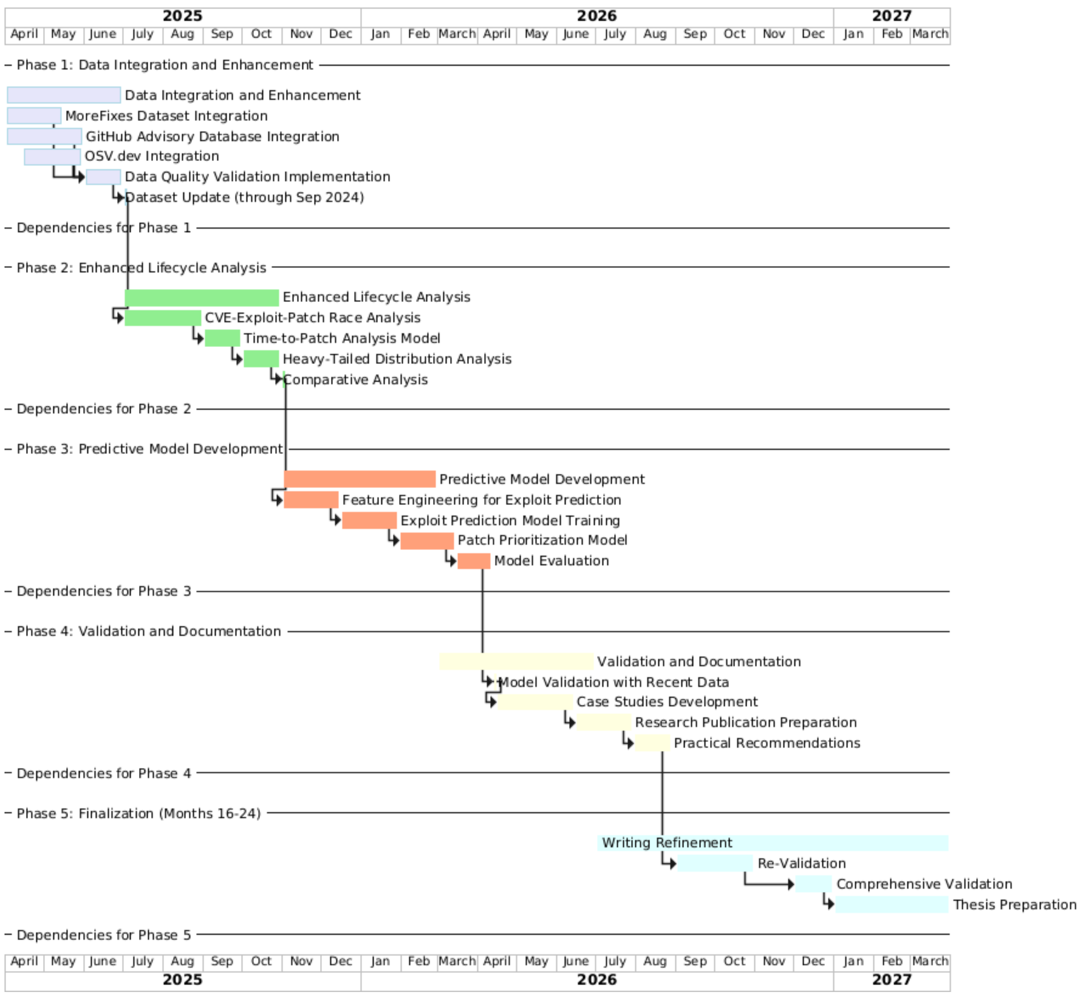


FIGURE 6.3: Research Timeline Gantt Chart

### 6.4.1 Phase 1: Data Integration and Enhancement (Months 1-3)

The initial phase will focus on expanding our data sources to provide a more comprehensive foundation for subsequent analysis:

- **MoreFixes Dataset Integration (1.5 months, March - mid-April 2025)**: We will begin with integrating the MoreFixes dataset [4], which contains 26,617 unique CVEs with fix commits from 6,945 GitHub projects. This will significantly expand our view of patch availability beyond Microsoft products.
- **GitHub Advisory Database Integration (2 months, March - April 2025)**: We will develop matching algorithms to correlate GitHub Advisory entries with our existing CVE dataset, focusing particularly on enhancing our understanding of exploited vulnerabilities.
- **OSV.dev Integration (1.5 months, mid-April - May 2025)**: Starting mid-April 2025, we will integrate data from OSV.dev, with particular focus on vulnerabilities in open-source components that interact with Microsoft products, addressing the challenges identified in Section 5.4.1 regarding third-party components.
- **Data Quality Validation Implementation (2 months, May - June 2025)**: Beginning May 2025, we will implement enhanced validation mechanisms following

the approach outlined in MoreFixes [4], which uses a blocklist to filter out irrelevant repositories and a scoring system to evaluate the relevance of potential fix commits.

- **Dataset Update (1 month, June 2025)**: In June 2025, we will update our existing datasets to include the most recent vulnerabilities through September 2024.

### 6.4.2 Phase 2: Enhanced Lifecycle Analysis (Months 4-7)

The second phase will focus on developing and applying the enhanced analytical approaches described in Section 6.2:

- **CVE-Exploit-Patch Race Analysis (2 months, July - August 2025)**: We will develop statistical models to characterize the three-way race between CVE reservation, exploit development, and patch deployment, extending our findings from Section 5.3.1.3. This will leverage the expanded dataset from Phase 1 to provide more robust statistical analysis.
- **Time-to-Patch Analysis Model (2 months, August - September 2025)**: Beginning August 2025, we will implement detailed analysis of patching timelines, comparing Microsoft's patterns with those of other vendors represented in the MoreFixes dataset. This directly builds on our findings in Section 5.3.1.2 regarding the time between exploit publication and patch availability.
- **Heavy-Tailed Distribution Analysis (2 months, September - October 2025)**: Starting September 2025, we will apply heavy-tailed distribution models to identify characteristics of highly-exploited vulnerabilities, extending our analysis from Section 4.2.6 where we identified CVEs with high numbers of exploits.
- **Comparative Analysis (1 month, October 2025)**: In October 2025, we will conduct comparative analysis of lifecycle patterns across different vendors and product types, taking advantage of the broader coverage provided by the MoreFixes dataset.

### 6.4.3 Phase 3: Predictive Model Development (Months 8-11)

The third phase will focus on developing and evaluating the machine learning models described in Section 6.3:

- **Feature Engineering for Exploit Prediction (1.5 months, November - mid-December 2025)**: We will extract relevant features from vulnerability descriptions, CWE information, affected products, and severity scores, building on our findings from Chapter 4 regarding the factors associated with exploitation.
- **Exploit Prediction Model Training (1.5 months, mid-December 2025 - January 2026)**: Beginning mid-December 2025, we will train and validate the XGBoost model described in Section 6.3.1, using our expanded dataset that includes both Microsoft-related exploits and the broader exploitation data available through MoreFixes.
- **Patch Prioritization Model (1.5 months, January - mid-February 2026)**: Starting January 2026, we will develop the supervised learning model described in

Section 6.3.2, incorporating insights from our lifecycle analysis in Phase 2. This model will directly address the challenge identified in Section 5.3.1.2 regarding timely patching of critical vulnerabilities.

- **Model Evaluation (1 month, mid-February - March 2026)**: In mid-February 2026, we will evaluate both models using historical data, with a focus on their performance for high-severity vulnerabilities as identified in Section 5.3.2.

### 6.4.4 Phase 4: Validation and Documentation (Months 12-15)

The fourth phase will focus on validating our results and preparing comprehensive documentation:

- **Model Validation with Recent Data (1.5 months, March - mid-April 2026)**: We will validate our predictive models against recent vulnerabilities (post-September 2024) that were not included in the training data, providing a realistic assessment of their performance in real-world scenarios.
- **Case Studies Development (2 months, April - May 2026)**: Beginning April 2026, we will develop detailed case studies focusing on specific vulnerabilities that exemplify the patterns identified in our analysis. These will include examples from both Microsoft products and other platforms represented in the MoreFixes dataset.
- **Research Publication Preparation (1.5 months, May - mid-June 2026)**: Starting May 2026, we will prepare journal and conference publications based on our findings, focusing particularly on the novel insights gained from our integrated dataset and predictive models.
- **Practical Recommendations (1 month, mid-June - July 2026)**: In mid-June 2026, we will develop concrete recommendations for vulnerability management practices based on our results, targeted specifically at enhancing the security of Microsoft products and their interactions with third-party components.

### 6.4.5 Phase 5: Finalization (Months 16-24)

The final phase focuses on refining our research, conducting final validations, and preparing the thesis:

- **Writing Refinement (9 months, July 2026 - March 2027)**: We will refine our research documents, improving clarity and coherence while incorporating feedback from previous phases.
- **Re-Validation (2 months, September - October 2026)**: Based on peer feedback, we will re-evaluate our models and findings, making necessary adjustments to improve accuracy and reliability.
- **Comprehensive Validation (1 month, December 2026)**: A final round of validation will be performed, testing our methodologies and models on diverse datasets to confirm their generalizability and robustness.
- **Thesis Preparation (3 months, January - March 2027)**: The last stage will focus on compiling all findings, methodologies, and results into a well-structured doctoral thesis that contributes to the field of vulnerability management.

This structured 24-month research plan ensures systematic progress toward addressing the identified research gaps while maintaining focus on practical outcomes. The timeline allows for iterative refinement of developed solutions while ensuring rigorous validation of results and sufficient time for documentation and dissemination of findings.

# APPENDIX A

## Exploits and CVE Description comparison between Microsoft MSRC DB and Exploit DB for CVE-2012-1199

**1. CVE Database Description:** BASE_path parameter to base_ag_main.php
**Exploit DB Description:** BASE 1.4.5 - 'base_ag_main.php?base_path' Remote File Inclusion
Matched

**2. CVE Database Description:** base_db_setup.php
**Exploit DB Description:** BASE 1.4.5 - 'base_db_setup.php?base_path' Remote File Inclusion
Matched

**3. CVE Database Description:** base_graph_common.php
**Exploit DB Description:** BASE 1.4.5 - 'base_graph_common.php?base_path' Remote File Inclusion
Matched

**4. CVE Database Description:** base_graph_display.php
**Exploit DB Description:** BASE 1.4.5 - 'base_graph_display.php?base_path' Remote File Inclusion
Matched

**5. CVE Database Description:** base_graph_form.php
**Exploit DB Description:** BASE 1.4.5 - 'base_graph_form.php?base_path' Remote File Inclusion
Matched

**6. CVE Database Description:** base_graph_main.php
**Exploit DB Description:** BASE 1.4.5 - 'base_graph_main.php?base_path' Remote File Inclusion
Matched

**7. CVE Database Description:** base_local_rules.php
**Exploit DB Description:** BASE 1.4.5 - 'base_local_rules.php?base_path' Remote File Inclusion
Matched

**8. CVE Database Description:** base_logout.php
**Exploit DB Description:** BASE 1.4.5 - 'base_logout.php?base_path' Remote File Inclusion
Matched

**9. CVE Database Description:** base_main.php
**Exploit DB Description:** BASE 1.4.5 - 'base_main.php?base_path' Remote File Inclusion
Matched

**10. CVE Database Description:** base_maintenance.php
**Exploit DB Description:** BASE 1.4.5 - 'base_maintenance.php?base_path' Remote File Inclusion
Matched
**11. CVE Database Description:** base_payload.php
**Exploit DB Description:** BASE 1.4.5 - 'base_payload.php?base_path' Remote File Inclusion
Matched
**12. CVE Database Description:** base_qry_alert.php
**Exploit DB Description:** BASE 1.4.5 - 'base_qry_alert.php?base_path' Remote File Inclusion
Matched
**13. CVE Database Description:** base_qry_common.php
**Exploit DB Description:** BASE 1.4.5 - 'base_qry_common.php?base_path' Remote File Inclusion
Matched
**14. CVE Database Description:** base_qry_main.php
**Exploit DB Description:** BASE 1.4.5 - 'base_qry_main.php?base_path' Remote File Inclusion
Matched
**15. CVE Database Description:** base_stat_alerts.php
**Exploit DB Description:** BASE 1.4.5 - 'base_stat_alerts.php?base_path' Remote File Inclusion
Matched
**16. CVE Database Description:** base_stat_class.php
**Exploit DB Description:** BASE 1.4.5 - 'base_stat_class.php?base_path' Remote File Inclusion
Matched
**17. CVE Database Description:** base_stat_common.php
**Exploit DB Description:** BASE 1.4.5 - 'base_stat_common.php?base_path' Remote File Inclusion
Matched
**18. CVE Database Description:** base_stat_ipaddr.php
**Exploit DB Description:** BASE 1.4.5 - 'base_stat_ipaddr.php?base_path' Remote File Inclusion
Matched
**19. CVE Database Description:** base_stat_iplink.php
**Exploit DB Description:** BASE 1.4.5 - 'base_stat_iplink.php?base_path' Remote File Inclusion
Matched
**20. CVE Database Description:** base_stat_ports.php
**Exploit DB Description:** BASE 1.4.5 - 'base_stat_ports.php?base_path' Remote File Inclusion
Matched
**21. CVE Database Description:** base_stat_sensor.php
**Exploit DB Description:** BASE 1.4.5 - 'base_stat_sensor.php?base_path' Remote File Inclusion
Matched
**22. CVE Database Description:** base_stat_time.php
**Exploit DB Description:** BASE 1.4.5 - 'base_stat_time.php?base_path' Remote File

Inclusion
Matched
**23. CVE Database Description:** base_stat_uaddr.php
**Exploit DB Description:** BASE 1.4.5 - 'base_stat_uaddr.php?base_path' Remote File Inclusion
Matched
**24. CVE Database Description:** base_user.php
**Exploit DB Description:** BASE 1.4.5 - 'base_user.php?base_path' Remote File Inclusion
Matched
**25. CVE Database Description:** index.php
**Exploit DB Description:** BASE 1.4.5 - 'index.php?base_path' Remote File Inclusion
Matched
**26. CVE Database Description:** admin/base_roleadmin.php
**Exploit DB Description:**
No Match
**27. CVE Database Description:** admin/base_useradmin.php
**Exploit DB Description:** BASE 1.4.5 - 'admin/base_useradmin.php?base_path' Remote File Inclusion
Matched
**28. CVE Database Description:** admin/index.php
**Exploit DB Description:** BASE 1.4.5 - 'admin/index.php?base_path' Remote File Inclusion
Matched
**29. CVE Database Description:** help/base_setup_help.php
**Exploit DB Description:** BASE 1.4.5 - 'help/base_setup_help.php?base_path' Remote File Inclusion
Matched
**30. CVE Database Description:** includes/base_action.inc.php
**Exploit DB Description:** BASE 1.4.5 - 'includes/base_action.inc.php?base_path' Remote File Inclusion
Matched
**31. CVE Database Description:** includes/base_cache.inc.php
**Exploit DB Description:** BASE 1.4.5 - 'includes/base_cache.inc.php?base_path' Remote File Inclusion
Matched
**32. CVE Database Description:** includes/base_db.inc.php
**Exploit DB Description:** BASE 1.4.5 - 'includes/base_db.inc.php?base_path' Remote File Inclusion
Matched
**33. CVE Database Description:** includes/base_include.inc.php
**Exploit DB Description:** BASE 1.4.5 - 'includes/base_include.inc.php?base_path' Remote File Inclusion
Matched
**34. CVE Database Description:** includes/base_output_html.inc.php
**Exploit DB Description:** BASE 1.4.5 - 'includes/base_output_html.inc.php?base_path' Remote File Inclusion
Matched
**35. CVE Database Description:** includes/base_output_query.inc.php
**Exploit DB Description:** BASE 1.4.5 - 'includes/base_output_query.inc.php?base_path'

Remote File Inclusion
Matched

**36. CVE Database Description:** includes/base_state_criteria.inc.php
**Exploit DB Description:** BASE 1.4.5 - 'includes/base_state_criteria.inc.php?base_path' Remote File Inclusion
Matched

**37. CVE Database Description:** includes/base_state_query.inc.php
**Exploit DB Description:** BASE 1.4.5 - 'includes/base_state_query.inc.php?base_path' Remote File Inclusion
Matched

**38. CVE Database Description:** setup/base_conf_contents.php
**Exploit DB Description:** BASE 1.4.5 - 'setup/base_conf_contents.php?base_path' Remote File Inclusion
Matched

**39. CVE Database Description:** includes/base_state_common.inc.php
**Exploit DB Description:** BASE 1.4.5 - 'includes/base_state_common.inc.php?base_path' Remote File Inclusion
Matched

**40. CVE Database Description:** GLOBALS[user_session_path] parameter to includes/base_state_common.inc.php
**Exploit DB Description:** BASE 1.4.5 - 'includes/base_state_common.inc.php? GLOBALS[user_session_path]' Remote File Inclusion
Matched

**41. CVE Database Description:** BASE_Language parameter to setup/base_conf_contents.php
**Exploit DB Description:** BASE 1.4.5 - 'setup/base_conf_contents.php?BASE_Language' Remote File Inclusion
Matched

**42. CVE Database Description:** ado_inc_php parameter to setup/setup2.php
**Exploit DB Description:** BASE 1.4.5 - 'setup/setup2.php?ado_inc_PHP' Remote File Inclusion
Matched

# APPENDIX B

# VULNERABILITIES WITH HIGH DIFFERENCES BETWEEN LIFETIME EVENTS *(More than 1000 days)*

#### B.0.0.1 Total of 9 Vulnerabilities for differences between Patching and Exploitation:

CVE-2016-6664, CVE-2009-4487, CVE-2013-0221, CVE-1999-0236, CVE-2016-7567, CVE-2009-4484, CVE-2010-2891, CVE-2019-15126, CVE-1999-1412

#### B.0.0.2 Total of 1 Vulnerability for differences between Exploitation and CVE creation:

CVE-2017-0148. *Note: This CVE is has also more than 1000 days difference between Patching and Exploitation*

# APPENDIX C

VULNERABILITIES WITH HIGH NUMBER OF EXPLOITS *(More than 13 exploits)*

## C.1 The below are examples of CVEs that holds more than 13 exploits *Same Characteristics*

- **CVE-2004-0067**, all characteristics in exploitDB are identical except for description. This cve describes multiple Cross-Site Scripting (XSS) vulnerabilities in phpGedView before version 2.65. These vulnerabilities allow remote attackers to inject arbitrary HTML or web scripts through various PHP files in the application. The vulnerabilities listed in your data are all related to this CVE and exhibit similarities in terms of the type of vulnerability and the affected files. We Compare it with its CVE Description in CVE V5 database. The CVE description for CVE-2004-0067 aligns closely with the exploits listed. It mentions multiple files vulnerable to XSS attacks, such as descendancy.php, index.php, individual.php, login.php, relationship.php, source.php, imageview.php, calendar.php, gedrecord.php, login.php, and gdbi_interface.php. Each file listed in the CVE description corresponds to the exploits in the provided dataset, validating the scope of the vulnerabilities. The dataset includes the same files mentioned in the CVE description, along with additional exploits like SQL Injection vulnerabilities in placelist.php and timeline.php, which might not be covered in the CVE but reflect broader issues in the same application. The consistency in file names and the nature of the vulnerabilities (XSS and SQL Injection) affirm that these exploits are relevant to the CVE-2004-0067 vulnerabilities.

- **CVE-2004-1925**, all characteristics in exploitDB are identical except for description. Both the descriptions and the CVE mention SQL injection vulnerabilities that arise due to unsanitized sort_mode and offset parameters. The affected files listed in the descriptions align with those mentioned in the CVE. They include:

  - tiki-usermenu.php
  - tiki-list_file_gallery.php
  - tiki-directory_ranking.php
  - tiki-browse_categories.php
  - tiki-file_galleries.php
  - tiki-list_faqs.php
  - tiki-list_trackers.php

- tiki-list_blogs.php
- tiki-index.php
- tiki-user_tasks.php
- tiki-directory_search.php

The CVE indicates issues with both sort_mode and offset parameters. These are reflected in the provided descriptions. The MITRE CVE description is more comprehensive and includes all vulnerable parameters and files. It confirms that vulnerabilities exist in both sort_mode and offset parameters across multiple PHP scripts, while the provided descriptions focus primarily on specific parameters and files but do not provide a complete overview of all affected files and parameters. The provided descriptions accurately reflect the vulnerabilities listed in CVE-2004-1925, focusing on specific instances of SQL injection vulnerabilities related to sort_mode and offset parameters. The CVE description includes a comprehensive list of all affected files and parameters, while the provided descriptions cover individual cases without explicitly mapping out the complete list. Both the descriptions and the CVE share the same underlying vulnerabilities related to SQL injection. The CVE provides a complete overview of affected parameters and files, while the individual descriptions match specific cases of these vulnerabilities.

## C.2 The below are examples of CVEs that have between 10 and 13 exploits *61% same Characteristics*

The purpose of this analysis is to identify differences in exploit characteristics, including author names, types, platforms, and descriptions. This information can provide valuable insights into the diversity of exploit development and the various ways vulnerabilities are targeted.

### C.2.1 Methodology

For each CVE, we examined:

- The original vulnerability description
- Associated exploits
- Exploit authors
- Exploit types
- Target platforms
- Exploit descriptions

We then compared these characteristics across exploits for each CVE to identify notable differences or patterns.

## C.3 Analysis

#### C.3.0.1 CVE-2000-0844

**Description:** Vulnerability in Unix locale subsystem allowing arbitrary command execution via functions like gettext and catopen.

| Characteristic | Analysis |
|---|---|
| Number of exploits | 11 |
| Authors | Various (Kil3r of Lam3rZ, localcore, sk8, logikal, warning3, Solar Eclipse, Synnergy.net, anonymous) |
| Types | All are local exploits |
| Platforms | Varied (immunix, linux, solaris, unix) |
| Descriptions | Mostly consistent, targeting the locale subsystem vulnerability on different Unix-like systems |

**Differences:**

- Multiple authors contributed exploits
- Platforms vary, covering different Unix-like systems
- Some exploits are more specific (e.g., targeting 'eject' on Solaris)

#### C.3.0.2 CVE-2002-1230

**Description:** NetDDE Agent vulnerability allowing arbitrary code execution via WM_COPYDATA and WM_TIMER messages.

| Characteristic | Analysis |
|---|---|
| Number of exploits | 10 |
| Authors | Varied (Serus, sectroyer, Oliver Lavery, Brett Moore, Ovidio Mallo, anonymous) |
| Types | All are local exploits |
| Platform | All target Windows |
| Descriptions | Consistent focus on the Window Message Subsystem Design Error |

**Differences:**

- Multiple authors contributed exploits
- Some exploits are more specific (e.g., targeting NetDDE specifically)

#### C.3.0.3 CVE-2003-0201

**Description:** Buffer overflow in Samba's call_trans2open function allowing remote code execution.

| Characteristic | Analysis |
|---|---|
| Number of exploits | 13 |

| | |
|---|---|
| Authors | Varied (Metasploit, Schizoprenic, H D Moore, eSDee, Xpl017Elz, c0wboy, eDSee, noir) |
| Types | All are remote exploits |
| Platforms | Varied (BSD, Linux, OSX, Solaris, Unix) |
| Descriptions | Consistent focus on the trans2open vulnerability |

**Differences:**

- Multiple authors contributed exploits
- Platforms vary, covering different operating systems
- Some exploits are more specific (e.g., targeting specific OS versions or architectures)

#### C.3.0.4 CVE-2003-0605

**Description:** RPC DCOM interface vulnerability in Windows 2000 allowing denial of service and privilege escalation.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 11 |
| Authors | Varied (Flashsky, pHrail, anonymous, oc192, Doke Scott, H D Moore, ins1der) |
| Types | Mostly remote, one DoS |
| Platform | All target Windows |
| Descriptions | Consistent focus on the RPC DCOM vulnerability |

**Differences:**

- Multiple authors contributed exploits
- Most exploits are remote, but one is specifically for DoS
- Some exploits target specific versions (e.g., Windows XP/2000)

#### C.3.0.5 CVE-2004-1924

**Description:** Multiple cross-site scripting vulnerabilities in Tiki CMS/Groupware.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 11 |
| Author | All by JeiAr |
| Type | All are web application exploits |
| Platform | All target PHP |
| Descriptions | Each exploit targets a specific vulnerable parameter or file |

**Differences:**

- Single author for all exploits
- Each exploit targets a different vulnerable parameter or file within the application

#### C.3.0.6 CVE-2006-0755

**Description:** Multiple PHP remote file inclusion vulnerabilities in dotProject.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 10 |
| Author | All by r.verton |
| Type | All are web application exploits |
| Platform | All target PHP |
| Descriptions | Each exploit targets a specific vulnerable file or parameter |

**Differences:**

- Single author for all exploits
- Each exploit targets a different vulnerable file or parameter within the application

#### C.3.0.7 CVE-2006-1205

**Description:** Multiple cross-site scripting vulnerabilities in myWebland myBloggie.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 10 |
| Author | All by enji@infosys.tuwien.ac.at |
| Type | All are web application exploits |
| Platform | All target PHP |
| Descriptions | Each exploit targets a specific vulnerable file or parameter |

**Differences:**

- Single author for all exploits
- Each exploit targets a different vulnerable file or parameter within the application

#### C.3.0.8 CVE-2006-3019

**Description:** Multiple PHP remote file inclusion vulnerabilities in phpCMS.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 10 |
| Author | All by Federico Fazzi |
| Type | All are web application exploits |
| Platform | All target PHP |
| Descriptions | Each exploit targets a specific vulnerable file |

**Differences:**

- Single author for all exploits
- Each exploit targets a different vulnerable file within the application

#### C.3.0.9 CVE-2006-4477

**Description:** Multiple PHP remote file inclusion vulnerabilities in Visual Shapers ezContents.

| Characteristic | Analysis |
|---|---|
| Number of exploits | 10 |
| Author | All by DarkFig |
| Type | All are web application exploits |
| Platform | All target PHP |
| Descriptions | Each exploit targets a specific vulnerable file or parameter |

**Differences:**

- Single author for all exploits
- Each exploit targets a different vulnerable file or parameter within the application

#### C.3.0.10 CVE-2006-5829

**Description:** Multiple SQL injection vulnerabilities in All In One Control Panel (AIOCP).

| Characteristic | Analysis |
|---|---|
| Number of exploits | 12 |
| Author | All by laurent gaffie |
| Type | All are web application exploits |
| Platform | All target PHP |
| Descriptions | Each exploit targets a specific vulnerable file or parameter |

**Differences:**

- Single author for all exploits
- Each exploit targets a different vulnerable file or parameter within the application

#### C.3.0.11 CVE-2007-0364

**Description:** Multiple cross-site scripting vulnerabilities in INDEXU.

| Characteristic | Analysis |
|---|---|
| Number of exploits | 13 |
| Author | All by SwEET-DeViL |
| Type | All are web application exploits |
| Platform | All target PHP |
| Descriptions | Each exploit targets a specific vulnerable file or parameter |

**Differences:**

- Single author for all exploits
- Each exploit targets a different vulnerable file or parameter within the application

#### C.3.0.12 CVE-2007-3217

**Description:** Multiple PHP remote file inclusion vulnerabilities in Prototype of an PHP application.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 13 |
| Author | All by pito pito |
| Type | All are web application exploits |
| Platform | All target PHP |
| Descriptions | Each exploit targets a specific vulnerable file or parameter |

**Differences:**

- Single author for all exploits
- Each exploit targets a different vulnerable file or parameter within the application

#### C.3.0.13 CVE-2007-5243

**Description:** Multiple stack-based buffer overflows in Borland InterBase.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 13 |
| Authors | Varied (Metasploit, Adriano Lima) |
| Types | All are remote exploits |
| Platforms | Varied (Linux, Windows) |
| Descriptions | Consistent focus on different buffer overflow vulnerabilities in InterBase functions |

**Differences:**

- Multiple authors contributed exploits
- Exploits target different vulnerable functions within InterBase
- Some exploits are specific to certain OS platforms

#### C.3.0.14 CVE-2008-3260

**Description:** Multiple cross-site scripting vulnerabilities in Claroline.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 12 |
| Author | All by Digital Security Research Group |

| | |
|---|---|
| Type | All are web application exploits |
| Platform | All target PHP |
| Descriptions | Each exploit targets a specific vulnerable file or parameter |

**Differences:**

- Single author for all exploits
- Each exploit targets a different vulnerable file or parameter within the application

#### C.3.0.15 CVE-2008-6840

**Description:** Multiple PHP remote file inclusion vulnerabilities in V-webmail.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 13 |
| Author | All by CraCkEr |
| Type | All are web application exploits |
| Platform | All target PHP |
| Descriptions | Each exploit targets a specific vulnerable file or parameter |

**Differences:**

- Single author for all exploits
- Each exploit targets a different vulnerable file or parameter within the application

#### C.3.0.16 CVE-2009-0689

**Description:** Array index error in dtoa implementation allowing denial of service and possible code execution.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 13 |
| Authors | Varied (Maksymilian Arciemowicz, sp3x, Alin Rad Pop) |
| Types | Mostly DoS, some remote code execution |
| Platforms | Varied (BSD, Linux, OSX, Windows, Multiple) |
| Descriptions | Consistent focus on the dtoa vulnerability, but targeting different applications and systems |

**Differences:**

- Multiple authors contributed exploits
- Exploits target different applications and systems affected by the vulnerability
- Some exploits focus on DoS, while others achieve remote code execution

#### C.3.0.17 CVE-2009-1330

**Description:** Stack-based buffer overflow in Easy RM to MP3 Converter.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 10 |
| Authors | Varied (Cyber-Zone, Oh Yaw Theng, bibi-info, d3b4g, Stack, Fitzl Csaba) |
| Types | Mostly local, some PoC |
| Platform | All target Windows |
| Descriptions | Consistent focus on buffer overflow in various file formats |

**Differences:**

- Multiple authors contributed exploits
- Some exploits are proof-of-concept, while others are fully developed
- Exploits target different file formats and converter applications

#### C.3.0.18 CVE-2009-3789

**Description:** Multiple cross-site scripting vulnerabilities in OpenDocMan.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 13 |
| Author | All by Amol Naik |
| Type | All are web application exploits |
| Platform | All target PHP |
| Descriptions | Each exploit targets a specific vulnerable file or parameter |

**Differences:**

- Single author for all exploits
- Each exploit targets a different vulnerable file or parameter within the application

#### C.3.0.19 CVE-2010-4120

**Description:** Multiple cross-site scripting vulnerabilities in IBM Tivoli Access Manager for e-business.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 11 |
| Author | All by IBM |
| Type | All are web application exploits |
| Platform | All target multiple platforms |
| Descriptions | Each exploit targets a specific vulnerable file or parameter |

**Differences:**

- Single author (IBM) for all exploits
- Each exploit targets a different vulnerable file or parameter within the application

#### C.3.0.20 CVE-2013-0135

**Description:** Multiple SQL injection vulnerabilities in PHP Address Book.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 12 |
| Author | All by Jurgen Voorneveld |
| Type | All are web application exploits |
| Platform | All target PHP |
| Descriptions | Each exploit targets a specific vulnerable file or parameter |

**Differences:**

- Single author for all exploits
- Each exploit targets a different vulnerable file or parameter within the application

#### C.3.0.21 CVE-2013-4730

**Description:** Buffer overflow in PCMan's FTP Server allowing remote code execution.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 11 |
| Authors | Varied (Chako, R-73eN, Koby, MSJ, Jacob Holcomb, Mahmod Mahajna, Ottomatik, Polunchis, Rick Flores, Sumit) |
| Type | All are remote exploits |
| Platform | All target Windows |
| Descriptions | Consistent focus on buffer overflow, but targeting different FTP commands |

**Differences:**

- Multiple authors contributed exploits
- Exploits target different FTP commands (e.g., USER, MKD, RENAME, ABOR, CWD, PASS, STOR)
- Some exploits are more specific or use different techniques (e.g., Metasploit module)

#### C.3.0.22 CVE-2014-1636

**Description:** Multiple SQL injection vulnerabilities in Command School Student Management System.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 12 |
| Author | All by AtT4CKxT3rR0r1ST |
| Type | All are web application exploits |
| Platform | All target PHP |
| Descriptions | Each exploit targets a specific vulnerable file or parameter |

**Differences:**

- Single author for all exploits
- Each exploit targets a different vulnerable file or parameter within the application

#### C.3.0.23 CVE-2018-14665

**Description:** Incorrect permission check in Xorg X11 server allowing privilege escalation.

| **Characteristic** | **Analysis** |
|---|---|
| Number of exploits | 11 |
| Authors | Varied (0xdono, bolonobolo, Metasploit, Marco Ivaldi, Hacker Fantastic) |
| Type | All are local privilege escalation exploits |
| Platforms | Varied (AIX, Linux, OpenBSD, Solaris, Unix, Multiple) |
| Descriptions | Consistent focus on the Xorg vulnerability, but with different approaches and target systems |

**Differences:**

- Multiple authors contributed exploits
- Exploits target different operating systems and environments
- Some exploits are more specific (e.g., targeting specific OS versions or using specific techniques)

We analyzed the above 23 Common Vulnerabilities and Exposures (CVEs) for consistency in author, type, platform, and description across their associated exploits, as well as similarity to the CVE description.

#### C.3.0.24 Analysis

CVEs with consistent characteristics (same author, same type, same platform) and descriptions that are similar across exploits and align closely with the CVE description:

1. CVE-2004-1924
2. CVE-2006-0755
3. CVE-2006-1205

4. CVE-2006-3019
5. CVE-2006-4477
6. CVE-2006-5829
7. CVE-2007-0364
8. CVE-2007-3217
9. CVE-2008-3260
10. CVE-2008-6840
11. CVE-2009-3789
12. CVE-2010-4120
13. CVE-2013-0135
14. CVE-2014-1636

CVEs that showed significant variations in author, type, platform, or description across exploits:

15. CVE-2000-0844
16. CVE-2002-1230
17. CVE-2003-0201
18. CVE-2003-0605
19. CVE-2007-5243
20. CVE-2009-0689
21. CVE-2009-1330
22. CVE-2013-4730
23. CVE-2018-14665

# APPENDIX D

VULNERABILITIES WITH HIGH NUMBER OF EXPLOITS

**The below are examples of CVE that holds more than 13 exploits and affect multiple software and hardwares**

## Summary of Exploits for CVE-2014-7910

The **CVE-2014-7910** vulnerability, also known as "Shellshock," is a high-severity security flaw in the GNU Bash (Bourne Again Shell). This vulnerability allows attackers to execute arbitrary commands on vulnerable systems by crafting malicious environment variables. Below is a summary of the 14 different exploits associated with **CVE-2014-7910**:

1. **2096**
   **Path:** exploits/cgi/remote/34777.rb
   **Name:** GNU Bash - Environment Variable Command Injection (Metasploit)
   **Published:** 2014-09-25
   **Author:** Shaun Colley
   **Type:** Remote
   **Platform:** CGI
   **References:** CVE-2014-7910, OSVDB-112004, CVE-2014-7227, CVE-2014-7196, CVE-2014-7169, CVE-2014-62771, CVE-2014-6271, CVE-2014-3671, CVE-2014-3659

2. **2280**
   **Path:** exploits/cgi/webapps/34895.rb
   **Name:** Bash CGI - 'Shellshock' Remote Command Injection (Metasploit)
   **Published:** 2014-10-06
   **Author:** Fady Mohammed Osman
   **Type:** Webapps
   **Platform:** CGI
   **References:** CVE-2014-7910, CVE-2014-7227, CVE-2014-7196, CVE-2014-7169, OSVDB-112004, CVE-2014-62771, CVE-2014-6271, CVE-2014-3671, CVE-2014-3659

3. **2482**
   **Path:** exploits/cgi/webapps/34839.py
   **Name:** IPFire - CGI Web Interface (Authenticated) Bash Environment Variable Code Injection
   **Published:** 2014-10-01
   **Author:** Claudio Viviani

**Type:** Webapps
**Platform:** CGI
**References:** CVE-2014-7910, CVE-2014-7227, CVE-2014-7196, CVE-2014-7169, CVE-2014-62771, OSVDB-112004, CVE-2014-6271, CVE-2014-3671, CVE-2014-3659

4. **3870**
**Path:** exploits/hardware/remote/36503.rb
**Name:** QNAP - Admin Shell via Bash Environment Variable Code Injection (Metasploit)
**Published:** 2015-03-26
**Author:** Patrick Pellegrino
**Type:** Remote
**Platform:** Hardware
**References:** CVE-2014-7910, OSVDB-112004, CVE-2014-7227, CVE-2014-7196, CVE-2014-7169, CVE-2014-62771, CVE-2014-6271, CVE-2014-3671, CVE-2014-3659

5. **3871**
**Path:** exploits/hardware/remote/36504.rb
**Name:** QNAP - Web Server Remote Code Execution via Bash Environment Variable Code Injection (Metasploit)
**Published:** 2015-03-26
**Author:** Patrick Pellegrino
**Type:** Remote
**Platform:** Hardware
**References:** CVE-2014-7910, OSVDB-112004, CVE-2014-7227, CVE-2014-7196, CVE-2014-7169, CVE-2014-62771, CVE-2014-6271, CVE-2014-3671, CVE-2014-3659

6. **8018**
**Path:** exploits/linux/remote/34766.php
**Name:** Bash - 'Shellshock' Environment Variables Command Injection
**Published:** 2014-09-25
**Authors:** Prakhar Prasad & Subho Halder
**Type:** Remote
**Platform:** Linux
**References:** CVE-2014-7910, OSVDB-112004, CVE-2014-7227, CVE-2014-7196, CVE-2014-7169, CVE-2014-62771, CVE-2014-6271, CVE-2014-3671, CVE-2014-3659

7. **8098**
**Path:** exploits/linux/remote/35115.rb
**Name:** CUPS Filter - Bash Environment Variable Code Injection (Metasploit)
**Published:** 2014-10-29
**Author:** Metasploit
**Type:** Remote
**Platform:** Linux
**References:** CVE-2014-7910, CVE-2014-7227, CVE-2014-7196, CVE-2014-7169, CVE-2014-62771, OSVDB-112004, CVE-2014-6271, CVE-2014-3671, CVE-2014-3659

8. **8211**
**Path:** exploits/linux/remote/34765.txt
**Name:** GNU Bash - 'Shellshock' Environment Variable Command Injection
**Published:** 2014-09-25
**Author:** Stephane Chazelas
**Type:** Remote
**Platform:** Linux
**References:** CVE-2014-7910, OSVDB-112004, CVE-2014-7227, CVE-2014-7196, CVE-2014-7169, CVE-2014-62771, CVE-2014-6271, CVE-2014-3671, CVE-2014-3659

9. **8212**
**Path:** exploits/linux/remote/34860.py
**Name:** GNU bash 4.3.11 - Environment Variable dhclient
**Published:** 2014-10-02
**Author:** @0x00string
**Type:** Remote
**Platform:** Linux
**References:** CVE-2014-7910, OSVDB-112169, CVE-2014-7227, CVE-2014-7196, CVE-2014-7187, CVE-2014-7186, CVE-2014-7169, CVE-2014-6278, CVE-2014-62771, OSVDB-112158, OSVDB-112097, OSVDB-112096, OSVDB-112004, CVE-2014-6277, CVE-2014-6271, CVE-2014-3671, CVE-2014-3659

10. **8547**
**Path:** exploits/linux/remote/34879.txt
**Name:** GNU bash - 'Shellshock' Remote Code Injection Vulnerability
**Published:** 2014-10-05
**Author:**
**Type:** Remote
**Platform:** Linux
**References:** CVE-2014-7910, CVE-2014-7227, CVE-2014-7196, CVE-2014-7169, OSVDB-112004, CVE-2014-62771, CVE-2014-6271, CVE-2014-3671, CVE-2014-3659

11. **8556**
**Path:** exploits/linux/remote/34895.py
**Name:** Bash - 'Shellshock' Remote Command Injection Vulnerability (dhclient)
**Published:** 2014-10-06
**Author:** William Bowling
**Type:** Remote
**Platform:** Linux
**References:** CVE-2014-7910, CVE-2014-7227, CVE-2014-7196, CVE-2014-7169, OSVDB-112004, CVE-2014-62771, CVE-2014-6271, CVE-2014-3671, CVE-2014-3659

12. **8558**
**Path:** exploits/linux/remote/34900.rb
**Name:** Linux BASH Remote Code Injection via Environment Variables (Shellshock) (Metasploit)
**Published:** 2014-10-06
**Author:** Metasploit
**Type:** Remote

**Platform:** Linux
**References:** CVE-2014-7910, CVE-2014-7227, CVE-2014-7196, CVE-2014-7169, OSVDB-112004, CVE-2014-62771, CVE-2014-6271, CVE-2014-3671, CVE-2014-3659

13. **8630**
    **Path:** exploits/linux/remote/34897.py
    **Name:** ProFTPd 1.3.5b - 'Shellshock' Remote Command Injection
    **Published:** 2014-10-06
    **Author:** Fr0st
    **Type:** Remote
    **Platform:** Linux
    **References:** CVE-2014-7910, CVE-2014-7227, CVE-2014-7196, CVE-2014-7169, OSVDB-112004, CVE-2014-62771, CVE-2014-6271, CVE-2014-3671, CVE-2014-3659

14. **8637**
    **Path:** exploits/linux/remote/34861.py
    **Name:** GNU bash - DHCP Client Bash Environment Variable Code Injection
    **Published:** 2014-10-02
    **Author:** Tim Brown
    **Type:** Remote
    **Platform:** Linux
    **References:** CVE-2014-7910, CVE-2014-7227, CVE-2014-7196, CVE-2014-7169, OSVDB-112004, CVE-2014-62771, CVE-2014-6271, CVE-2014-3671, CVE-2014-3659

## Explanation

These exploits exploit the "Shellshock" vulnerability in various environments and applications, including web servers, Linux systems, and hardware devices. The exploits target platforms like CGI, Linux, and hardware, using different payloads and methods to exploit the vulnerability in Bash. Proper patching and updating to a secure Bash version is critical to mitigate the risks associated with these exploits.

# APPENDIX E

## TOP REOCCURRING CWE OVER YEAR

| **CWE\Year** | **1999** | **2000** | **2001** | **2002** | **2003** | **2004** | **2005** | **2006** | **2007** | **2008** | **2009** | **2010** | **2011** | **2012** | **2013** | **2014** | **2015** | **2016** | **2017** | **2018** | **2019** | **2020** | **2021** | **2022** | **2023** | **2024** | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **CWE-79** | 0 | 0 | 0 | 2 | 3 | 3 | 48 | 115 | 389 | 764 | 655 | 397 | 344 | 523 | 411 | 925 | 640 | 470 | 1365 | 1738 | 2282 | 2164 | 2697 | 4803 | 4536 | 1451 | **26725** |
| **CWE-119** | 4 | 5 | 3 | 8 | 10 | 17 | 53 | 151 | 499 | 555 | 505 | 461 | 526 | 597 | 591 | 736 | 955 | 1040 | 2085 | 945 | 483 | 181 | 318 | 840 | 229 | 105 | **11902** |
| **CWE-89** | 0 | 0 | 0 | 1 | 3 | 1 | 51 | 97 | 275 | 1076 | 863 | 374 | 229 | 173 | 101 | 249 | 176 | 94 | 459 | 433 | 524 | 465 | 738 | 2370 | 1991 | 613 | **11356** |
| **CWE-20** | 1 | 10 | 4 | 4 | 5 | 8 | 40 | 46 | 259 | 372 | 285 | 246 | 313 | 280 | 341 | 530 | 377 | 536 | 1050 | 1054 | 918 | 835 | 668 | 1268 | 760 | 235 | **10445** |
| **CWE-787** | 0 | 0 | 0 | 0 | 1 | 7 | 1 | 5 | 8 | 5 | 10 | 23 | 27 | 30 | 13 | 13 | 12 | 171 | 251 | 817 | 1289 | 1388 | 1615 | 2252 | 1936 | 307 | **10181** |
| **CWE-200** | 3 | 8 | 3 | 5 | 3 | 1 | 19 | 30 | 116 | 189 | 139 | 120 | 154 | 148 | 164 | 334 | 580 | 691 | 1326 | 943 | 552 | 352 | 311 | 977 | 501 | 183 | **7852** |
| **CWE-22** | 0 | 0 | 2 | 2 | 0 | 4 | 16 | 27 | 185 | 349 | 303 | 220 | 78 | 84 | 53 | 177 | 132 | 78 | 267 | 505 | 473 | 436 | 536 | 1044 | 745 | 186 | **5902** |
| **CWE-125** | 0 | 0 | 0 | 0 | 0 | 5 | 1 | 0 | 1 | 0 | 1 | 1 | 25 | 16 | 4 | 9 | 4 | 88 | 654 | 694 | 902 | 672 | 729 | 988 | 907 | 163 | **5864** |
| **CWE-352** | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 3 | 38 | 75 | 98 | 53 | 42 | 103 | 58 | 222 | 214 | 84 | 282 | 373 | 532 | 401 | 467 | 1030 | 1153 | 395 | **5630** |
| **CWE-416** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 3 | 6 | 27 | 52 | 77 | 37 | 28 | 40 | 158 | 287 | 415 | 572 | 380 | 556 | 753 | 557 | 188 | **4139** |

TABLE E.1: Top 10 CWEs with affected CVEs yearly distribution

# APPENDIX F

# PREDICTIVE MODELING WITH MACHINE LEARNING

## F.1 Exploit Prediction Model

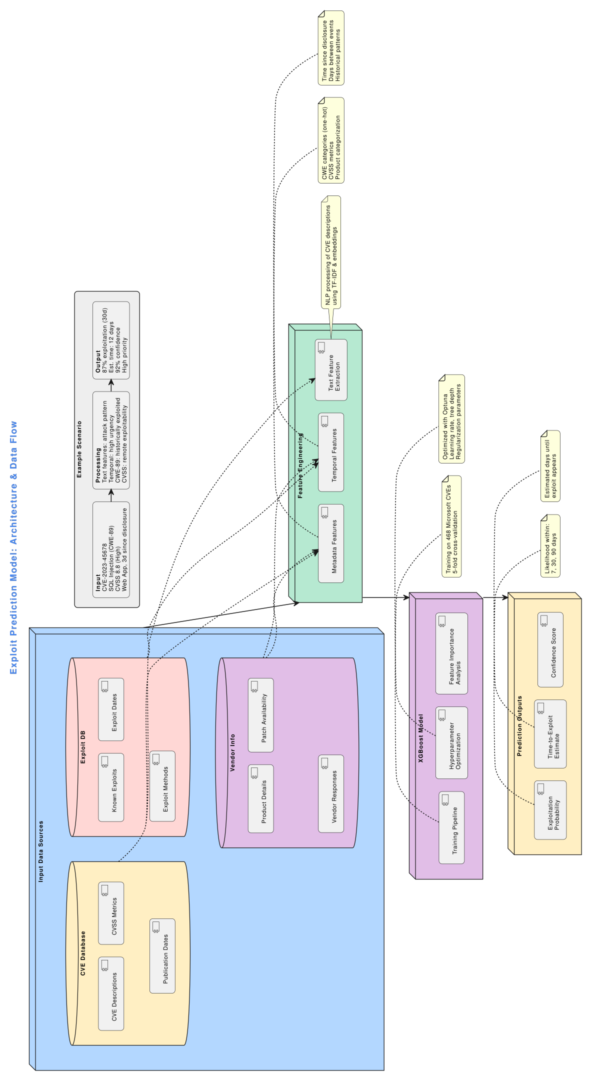


FIGURE F.1: Illustration of the exploitation prediction model.

## F.2 Patch Prioritization Model

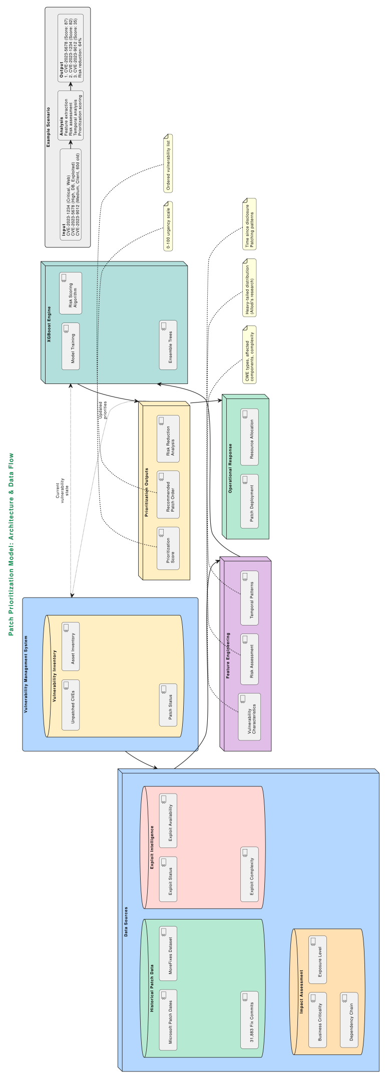


FIGURE F.2: Illustration of the patch prioritization model.

# BIBLIOGRAPHY

[1] S. Adam, *Unpatched vulnerabilities: The most brutal ransomware attack vector*, 2024. [Online]. Available: https://news.sophos.com/en-us/2024/04/03/unpatched-vulnerabilities-the-most-brutal-ransomware-attack-vector/.

[2] E. Aghaei and E. Al-Shaer, "Cve-driven attack technique prediction with semantic information extraction and a domain-specific language model," *arXiv preprint*, 2023. DOI: 10.48550/arxiv.2309.02785.

[3] V. Ahmadi, P. Arlos, and E. Casalicchio, "Normalization framework for vulnerability risk management in cloud," in *Conference on the Future of the Internet*, 2021, pp. 1–6. DOI: 10.1109/FICLOUD49777.2021.00022.

[4] J. Akhoundali, S. R. Nouri, and K. Rietveld, "Morefixes: A large-scale dataset of cve fix commits mined through enhanced repository discovery," in *Proceedings of Models and Data, 2024*, ACM, 2024. DOI: 10.1145/3663533.3664036. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3663533.3664036.

[5] T. Aladics, P. Hegedus, and R. Ferenc, "A comparative study of commit representations for jit vulnerability prediction," *Computers*, 2024. DOI: 10.3390/computers13010022.

[6] D. Alfasi, T. Shapira, and A. B. Barr, "Unveiling hidden links between unseen security entities," *arXiv preprint*, 2024. DOI: 10.48550/arxiv.2403.02014.

[7] A. Algarni, "The historical relationship between the software vulnerability lifecycle and vulnerability markets: Security and economic risks," *Computers*, 2022. DOI: 10.3390/computers11090137. [Online]. Available: https://doi.org/10.3390/computers11090137.

[8] L. Allodi, "The heavy tails of vulnerability exploitation," *IEEE Security & Privacy Magazine*, vol. 13, no. 2, pp. 70–76, Mar. 2015.

[9] L. Allodi, "Economic factors of vulnerability trade and exploitation: Empirical evidence from a prominent russian cybercrime market," *ACM Conference on Computer and Communications Security*, pp. 1483–1499, 2017. DOI: 10.1145/3133956.3133960.

[10] A. Anand, N. Bhatt, J. Kaur, and Y. Tamura, "Time lag-based modelling for software vulnerability exploitation process," *Journal of Cyber Security and Mobility*, pp. 663–678, 2021.

[11] R. Anderson, *Security engineering: A guide to building dependable distributed systems*. John Wiley & Sons, 2001.

[12] N. Antunes and M. Vieira, "On the metrics for benchmarking vulnerability detection tools," in *IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 505–516. DOI: 10.1109/DSN.2015.30.

[13] A. Anwar, A. Abusnaina, S. Chen, F. Li, and D. Mohaisen, "Cleaning the nvd: Comprehensive quality assessment, improvements, and analyses," *Dependable Systems and Networks*, 2021. DOI: 10.1109/DSN-S52858.2021.00011. [Online]. Available: https://doi.org/10.1109/DSN-S52858.2021.00011.

[14] I. Arce, "Vulnerabilities: Vulnerability management at the crossroads," *Network Security Archive*, 2008. DOI: 10.1016/S1353-4858(08)70064-3.

[15] A. Arora, R. Telang, and H. Xu, "Optimal policy for software vulnerability disclosure," *Management Science*, vol. 54, no. 4, pp. 642–656, 2008.

[16] S. A. Atiiq, C. Gehrmann, K. Dahl'en, and K. Khalil, "From generalist to specialist: Exploring cwe-specific vulnerability detection," *arXiv preprint*, 2024. DOI: 10.48550/arxiv.2408.02329.

[17] A. Avadanei, L. Nitescu, I. Constantin, and C. P. Sultanoiu, "Predictive model for software vulnerability management in telecommunication infrastructures," 2021. DOI: 10.1109/BLACKSEACOM52164.2021.9527768.

[18] M. H. Babu, N. S. N. S, M. Moharir, and M. Mohana, "Leveraging xgboost machine learning algorithm for common vulnerabilities and exposures (cve) exploitability classification," in *2024 International Conference on Computer Science, Information Technology and Smart Systems (CSITSS)*, 2024. DOI: 10.1109/csitss64042.2024.10816942.

[19] R. Bachmann and A. D. Brucker, "Developing secure software," *Datenschutz und Datensicherheit - DuD*, Mar. 2014.

[20] A. Balsam, M. Nowak, M. Walkowski, J. Oko, and S. Sujecki, "Analysis of cvss vulnerability base scores in the context of exploits' availability," *International Conference on Transparent Optical Networks*, 2023. DOI: 10.1109/icton59386.2023.10207394.

[21] D. Bassi and H. Singh, "A systematic literature review on software vulnerability prediction models," *IEEE Access*, vol. 11, pp. 98 765–98 789, 2023. DOI: 10.1109/access.2023.3312613.

[22] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, "A comparative analysis of gradient boosting algorithms," *Artificial Intelligence Review*, Mar. 2021. DOI: 10.1007/S10462-020-09896-5.

[23] M. Bishop, "A taxonomy of unix system and network vulnerabilities," Department of Computer Science, University of California at Davis, Tech. Rep. CSE-95-10, 1995.

[24] M. Bishop and D. Bailey, "A critical analysis of vulnerability taxonomies," Department of Computer Science at the University of California at Davis, Tech. Rep. CSE-96-11, 1996.

[25] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: Learning to classify vulnerabilities and predict exploits," *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 105–114, 2010. DOI: 10.1145/1835804.1835821.

[26] B. Brekkan, *Continue to safeguard your organization during nvd update delays*, Accessed: 2024-08-04. [Online]. Available: https://techcommunity.microsoft.com/t5/microsoft-defender-vulnerability/continue-to-safeguard-your-organization-during-nvd-update-delays/ba-p/4096409.

[27] M. Brilhante, D. Pestana, P. Pestana, and M. L. Rocha, "Measuring the risk of vulnerabilities exploitation," *AppliedMath*, vol. 4, pp. 20–54, Dec. 2023. DOI: 10.3390/appliedmath4010002. [Online]. Available: https://www.researchgate.net/publication/376802392_Measuring_the_Risk_of_Vulnerabilities_Exploitation.

[28] K. Carleton, "Recailibrating the use of zero-day vulnerabilities," *Journal of Law and Commerce*, vol. 42, no. 2, pp. 167–189, May 2024.

[29] B. A. Cheikes, D. Waltermire, and K. Scarfone, "Common platform enumeration: Naming specification version 2.3," *National Institute of Standards and Technology*, 2011.

[30] A. Cormack and É. Leverett, "Patchy incentives: Using law to encourage effective vulnerability response," *Journal of Cyber Policy*, vol. 8, no. 3, pp. 339–358, Dec. 2023.

[31] J. C. Costa, T. Roxo, J. B. Sequeiros, H. Proenca, and P. R. Inacio, "Predicting cvss metric via description interpretation," *IEEE Access*, vol. 10, pp. 59 125–59 134, 2022.

[32] R. Croft, M. A. Babar, and M. Kholoosi, "Data quality for software vulnerability datasets," *arXiv.Org*, 2023. DOI: 10.48550/arXiv.2301.05456. [Online]. Available: https://doi.org/10.48550/arXiv.2301.05456.

[33] Cybersecurity and I. S. Agency, *Ssvc-stakeholder-specific vulnerability categorization system*, Available online: https://www.cisa.gov/stakeholder-specific-vulnerability-categorization-ssvc (accessed on 8 November 2024), 2023.

[34] Cyentia Institute, "The evolving cve landscape," Cyentia Institute, Jan. 2023, Sponsored by F5 Labs. [Online]. Available: https://www.cyentia.com/cve-landscape/.

[35] H. K. Dam, T. Tran, T. Pham, S. W. Ng, J. Grundy, and A. Ghose, "Automatic feature learning for predicting vulnerable software components," *IEEE Transactions on Software Engineering*, vol. 47, no. 1, pp. 67–85, 2021.

[36] D. Dey, A. Lahiri, and G. Zhang, "Optimal policies for security patch management," *INFORMS Journal on Computing*, vol. 27, no. 3, pp. 462–477, 2015. DOI: 10.1287/IJOC.2014.0638.

[37] N. Dissanayake, M. Zahedi, A. Jayatilaka, and M. A. Babar, "Why, how and where of delays in software security patch management: An empirical investigation in the healthcare sector," *Proceedings of the ACM on Human-Computer Interaction*, vol. 6, no. CSCW2, 1âĂŞ29, Nov. 2022, ISSN: 2573-0142. DOI: 10.1145/3555087. [Online]. Available: http://dx.doi.org/10.1145/3555087.

[38] W. Du and A. P. Mathur, "Categorization of software errors that led to security breaches," in *Proceeding of the 21st National Information Systems Security Conference (NISSC'98)*, Crystal City, VA, 1998.

[39] D. G. Dzielski and T. Devine, “A systematic mapping study of the advancement in software vulnerability forecasting,” in *SoutheastCon*, 2023. DOI: `10.1109/SoutheastCon51012.2023.10115111`. [Online]. Available: `https://doi.org/10.1109/SoutheastCon51012.2023.10115111`.

[40] O. A. Ekle and D. Ulybyshev, “Enhanced categorization of cybersecurity vulnerabilities,” pp. 800–806, 2024. DOI: `10.1109/uemcon62879.2024.10754709`.

[41] M. W. Elbes, S. Hendawi, S. AlZu’bi, T. Kanan, and A. Mughaid, “Unleashing the full potential of artificial intelligence and machine learning in cybersecurity vulnerability management,” pp. 276–283, 2023. DOI: `10.1109/icit58056.2023.10225910`.

[42] M. Esposito and D. Falessi, “Validate: A deep dive into vulnerability prediction datasets,” *Information & Software Technology*, 2024. DOI: `10.1016/j.infsof.2024.107448`.

[43] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, “A c/c++ code vulnerability dataset with code changes and cve summaries,” in *Proceedings of the 17th International Conference on Mining Software Repositories*, ACM, 2020, pp. 508–512. DOI: `10.1145/3379597.3387501`.

[44] P. Foreman, *Vulnerability Management*. Boca Raton, FL: CRC Press, 2009.

[45] H. L. França, C Teixeira, and N. Laranjeiro, “Automating vulnerability management in the software development lifecycle,” *IEEE Access*, vol. 11, pp. 58 932–58 945, Jun. 2023.

[46] S. Frei, “Security econometrics the dynamics of (in)security,” *ETH ZURICH http://www.techzoom.net/publications*, 2009.

[47] T. Freitas, C. Novo, J. Soares, I. Dutra, M. E. Correia, B. Shariati, and R. Martins, “Hal 9000: A risk manager for itss,” *ArXiv*, 2024.

[48] T. Freitas, J. Soares, M. E. Correia, and R. Martins, “Skynet: A cyber-aware intrusion tolerant overseer,” in *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, IEEE, 2023, pp. 111–116.

[49] S. Furnell, “Vulnerability management,” *Network Security Archive*, 2016. DOI: `10.1016/S1353-4858(16)30036-8`.

[50] C. Gajiwala, “Artificial intelligence in cybersecurity : Advancing threat modeling and vulnerability assessment,” *International journal of scientific research in computer science, engineering and information technology*, 2024. DOI: `10.32628/cseit241051066`.

[51] R. K. Gandhi, A. Sharma, W. Mahoney, W. Soubramanien, and M. Jothilingam, “A systematic review of osint techniques, challenges and solutions in cybersecurity,” *ACM Computing Surveys*, vol. 54, no. 7, pp. 1–35, 2021. DOI: `10.1145/3465757`.

[52] M. Garcia, A. Bessani, and N. Neves, “Lazarus: Automatic management of diversity in bft systems,” in *Proceedings of the 20th International Middleware Conference*, ACM, 2019, pp. 241–254. DOI: `10.1145/3361525.3361555`.

[53] K. Garg, “How can artificial intelligence be used to detect and mitigate zero-day vulnerabilities?” *Security Research Quarterly*, vol. 12, no. 4, pp. 78–92, Oct. 2024.

[54] Google, *Osv.dev*, `https://osv.dev/`, Accessed: 2025-03-06.

[55] A. Gueye and P. Mell, "A historical and statistical study of the software vulnerability landscape," *arXiv: Cryptography and Security*, 2021.

[56] H. Guo, Z. Xing, and X. Li, "Predicting missing information of key aspects in vulnerability reports," *arXiv: Software Engineering*, 2020.

[57] K. Hanifi, R. F. Fouladi, B. G. Unsalver, and G. Karadag, "Software vulnerability prediction knowledge transferring between programming languages," *arXiv preprint*, 2023. DOI: 10.48550/arxiv.2303.06177.

[58] W. Hao, G. Linying, and Z. Nan, "Vulnerability management system and vulnerability management method," *International Journal of Network Security*, vol. 19, no. 5, pp. 851–860, 2017.

[59] M. Hassanali, M. Soltanaghaei, T. J. Gandomani, and F. Z. Boroujeni, "Software development effort estimation using boosting algorithms and automatic tuning of hyperparameters with optuna," *Journal of Software: Evolution and Process*, Apr. 2024. DOI: 10.1002/smr.2665.

[60] A. Hovsepyan, R. Scandariato, and W. Joosen, "Is newer always better?: The case of vulnerability prediction models," *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 1–6, 2016.

[61] M. Howard, D. LeBlanc, and J. Viega, "19 deadly sins of software security," 2005.

[62] W. Hu and V. L. L. Thing, "Cpe-identifier: Automated cpe identification and cve summaries annotation with deep learning and nlp," *arXiv preprint*, 2024. DOI: 10.48550/arxiv.2405.13568.

[63] Z. Hui, L. Liu, G. L. Chen, T. Cai, and C. T. Zhou, "Elevating security analysis: The mtlpt framework for enhanced vulnerability prediction," *Research Square Preprint*, 2024. DOI: 10.21203/rs.3.rs-4389278/v1.

[64] E. Iannone, R. Guadagni, F. Ferrucci, A. D. Lucia, and F. Palomba, "The secret life of software vulnerabilities: A large-scale empirical study," *IEEE Transactions on Software Engineering*, 2023. DOI: 10.1109/TSE.2022.3140868. [Online]. Available: https://doi.org/10.1109/TSE.2022.3140868.

[65] Ivanti, *71% of it security pros find patching to be overly complex and time consuming, ivanti study confirms*, Survey-2021, 2021. [Online]. Available: https://www.ivanti.com/company/press-releases/2021/71-of-it-security-pros-find-patching-to-be-overly-complex-and-time-consuming-ivanti-study-confirms.

[66] J. Jacobs, "Github: A source for exploits," *Cyentia Institute*, 2021. [Online]. Available: https://www.cyentia.com/github-a-source-for-exploits/.

[67] J. Jacobs, B. Romanosky, and Roytman, "Enhancing vulnerability prioritization: Data-driven exploit predictions with community-driven insights," arXiv preprint, 2023. [Online]. Available: https://arxiv.org/abs/2302.14172.

[68] J. Jacobs, S. Romanosky, B. Edwards, M. Roytman, and I. Adjerid, "Exploit prediction scoring system (epss)," in *Black Hat USA 2019*, 2019. [Online]. Available: https://arxiv.org/pdf/1908.04856.

[69] A. Jenkins, M. Wolters, and K. Vaniea, "To patch, or not to patch? that is the question: A case study of system administrators' online collaborative behaviour," *arXiv preprint*, 2023. DOI: 10.48550/arXiv.2307.03609.

[70] R. Johnson, A. Nappa, L. Bilge, J. Caballero, and T. Dumitras, "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, 2015, pp. 692–708.

[71] B. Jung, Y. Li, and T. Bechor, "Cavp: A context-aware vulnerability prioritization model," *Computers & Security*, vol. 122, p. 102 639, 2022. DOI: 10.1016/j.cose.2022.102639.

[72] M. L. S. JÃžnior, V. V. Cogo, and A. O. SÃą, "Secscore: Enhancing the cvss threat metric group with empirical evidences," *arXiv preprint*, 2024. DOI: 10.48550/arxiv.2405.08539.

[73] S. K. Kande, "Enhancing software vulnerability prediction models," *International Journal of Computing and Engineering*, 2024. DOI: 10.47941/ijce.2258.

[74] O. F. Keskin, N. Gannon, B. Lopez, and U. Tatar, "Scoring cyber vulnerabilities based on their impact on organizational goals," *Systems and Information Engineering Design Symposium*, 2021. DOI: 10.1109/SIEDS52267.2021.9483741.

[75] A. Khazaei, M. Ghasemzadeh, and V. Derhami, "An automatic method for cvss score prediction using vulnerabilities description," *Journal of Intelligent & Fuzzy Systems*, vol. 30, no. 1, pp. 89–96, 2016.

[76] J. Kim and Y. Won, "Patch integrity verification method using dual electronic signatures," *Journal of Information Processing Systems*, vol. 13, no. 6, pp. 1440–1447, 2017.

[77] H. K. R. Kommera, "Adaptive cybersecurity in the digital age: Emerging threat vectors and next-generation defense strategies," *International Journal For Science Technology And Engineering*, 2024. DOI: 10.22214/ijraset.2024.64226.

[78] V. V. Krishnan, "Ground zero: An in-depth analysis of 2022's zero-day vulnerabilities," *International Journal of Science and Research*, vol. 13, no. 9, pp. 234–249, Sep. 2024.

[79] I. V. Krsul, "Computer vulnerability analysis," Ph.D. dissertation, Purdue University, 1998.

[80] P. Kuehn, M. Bayer, M. Wendelborn, and C. Reuter, "Ovana: An approach to analyze and improve the information quality of vulnerability databases," *Availability, Reliability and Security*, 2021. DOI: 10.1145/3465481.3465744. [Online]. Available: https://doi.org/10.1145/3465481.3465744.

[81] R. L. Kumar and O. Temizkan, "Vendor response to software vulnerability: An economic investigation," *Decision Sciences*, vol. 43, no. 4, pp. 587–611, 2012. DOI: 10.1111/j.1540-5915.2012.00364.x.

[82] A. Kumari and I. Sharma, "Securing vulnerabilities: Fuzzer detection with machine learning classification," 2024, pp. 1–6. DOI: 10.1109/icaect60202.2024.10469056.

[83] H. S. Lallie, L. A. Shepherd, J. R. Nurse, A. Erola, G. Epiphaniou, C. Maple, and X. Bellekens, "Cyber security in the age of covid-19: A timeline and analysis of cyber-crime and cyber-attacks during the pandemic," *Computers amp; Security*, vol. 105, p. 102 248, Jun. 2021, ISSN: 0167-4048. DOI: 10.1016/j.cose.2021.102248. [Online]. Available: http://dx.doi.org/10.1016/j.cose.2021.102248.

[84] C. E. Landwehr, A. R. Bull, J. P. McDermott, and W. S. Choi, "A taxonomy of computer program security flaws," *ACM Computing Surveys*, vol. 26, no. 3, pp. 211–254, 1994.

[85] T. H. M. Le, X. Du, and M. A. Babar, "Are latent vulnerabilities hidden gems for software vulnerability prediction? an empirical study," *arXiv preprint*, 2024. DOI: 10.48550/arxiv.2401.11105.

[86] F. Li and V. Paxson, "A large-scale empirical study of security patches," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, New York, NY, USA: Association for Computing Machinery, 2017, pp. 2201–2215. DOI: 10.1145/3133956.3134072.

[87] X. Liao, K. Yuan, X. Wang, Z. Li, L. Xing, and R. Beyah, "Acing the ioc game: Toward automatic generation of indicators of compromise," *ACM Transactions on Privacy and Security*, vol. 19, no. 3, pp. 1–39, 2016.

[88] G. Limongi and A. Galderisi, "Twenty years of european and international research on vulnerability: A multi-faceted concept for better dealing with evolving risk landscapes," *International Journal of Disaster Risk Reduction*, 2021. DOI: 10.1016/J.IJDRR.2021.102451.

[89] P. Liu, Y. Bi, J. Huang, X. Jiang, and S. Wang, "Crepair: Cvae-based automatic vulnerability repair technology," *arXiv preprint*, 2024. DOI: 10.48550/arxiv.2411.05540.

[90] L. Maghrabi, E. Pfluegel, L. Al-Fagih, R. Graf, G. Settanni, and F. Skopik, "Improved software vulnerability patching techniques using cvss and game theory," in *International Conference on Cyber Security And Protection Of Digital Services*, 2017, pp. 1–6. DOI: 10.1109/CYBERSECPODS.2017.8074856.

[91] G. V. Marconato, M. Kaâniche, and V. Nicomette, "A vulnerability life cycle-based security modeling and evaluation approach," *The Computer Journal*, vol. 56, no. 4, pp. 422–442, 2013. DOI: 10.1093/comjnl/bxs112.

[92] G. Marconato, V. Nicomette, and M. Kaaniche, "Security-related vulnerability life cycle analysis," in *2012 7th International Conference on Risks and Security of Internet and Systems (CRiSIS)*, IEEE, Oct. 2012, pp. 1–8.

[93] V. Markkanen and T. Frantti, "Patch management planning - towards one-to-one policy," *IEEE Data Science and Analytics Conference*, 2023. DOI: 10.1109/dsa59317.2023.00018.

[94] R. A. Martin and S. Barnum, "Common weakness enumeration (cwe) status update," *Ada Letters*, vol. 27, no. 3, pp. 88–91, 2007.

[95] R. A. Martin, S. Barnum, and S. Christey, "Advancing software security through common weakness enumeration (cwe) and common attack pattern enumeration and classification (capec)," in *Proceedings of the 2007 Workshop on Software Security for Anti-Tamper Applications*, Software Engineering Institute, 2007, pp. 38–51. DOI: 10.1184/R1/6571826.v1.

[96] E. Maza and K. Z. Sultana, "Identifying evolution of software metrics by analyzing vulnerability history in open source projects," in *IEEE International Conference on Big Data, Cloud Computing, and Applications and Technologies*, 2022. DOI: 10.1109/BDCAT56447.2022.00039. [Online]. Available: https://doi.org/10.1109/BDCAT56447.2022.00039.

[97] V. A. Mehri, P. Arlos, and E. Casalicchio, "Automated context-aware vulnerability risk management for patch prioritization," *Electronics*, vol. 11, no. 21, p. 3580, 2022. DOI: 10.3390/electronics11213580.

[98] V. A. Mehri, P. Arlos, and E. Casalicchio, "Automated patch management: An empirical evaluation study," *IEEE Computer Security Foundations Symposium*, 2023. DOI: 10.1109/csr57506.2023.10224970.

[99] P. Mell, K. Kent, and J. Nusbaum, "Creating a patch and vulnerability management program," National Institute of Standards and Technology, Gaithersburg, MD, Special Publication 800-40 Version 2.0, 2007.

[100] P. Mell, K. Scarfone, and S. Romanosky, "The common vulnerability scoring system (cvss) and its applicability to federal agency systems," *National Institute of Standards and Technology*, 2006.

[101] Metasploit, *Metasploit framework*. [Online]. Available: https://www.metasploit.com/.

[102] P. Meunier, "Classes of vulnerabilities and attacks," *Wiley Handbook of Science and Technology for Homeland Security*, pp. 1–20, 2007.

[103] Microsoft, *Microsoft digital defense report 2022*, Accessed: 2025-03-09, 2022. [Online]. Available: https://www.microsoft.com/en-us/security/security-insider/intelligence-reports/microsoft-digital-defense-report-2022.

[104] Microsoft, "Microsoft security update guide dataset," 2024. [Online]. Available: https://msrc.microsoft.com/update-guide/.

[105] J. Miller and A. Smith, "Why, how and where of delays in software security patch management," *arXiv preprint arXiv:2202.09016*, 2022. [Online]. Available: https://arxiv.org/pdf/2202.09016.pdf.

[106] S. R. Mishra, V. M. Phuc, and N. V. Tanh, "Using security metrics to determine security program effectiveness," *AHFE International Conference*, 2023. DOI: 10.54941/ahfe1003720.

[107] MITRE, "Common weaknesses dataset," 2024. [Online]. Available: https://cwe.mitre.org/data/csv/1000.csv.zip.

[108] MITRE, "Cve v5 dataset," 2024. [Online]. Available: https://github.com/CVEProject/cvelistV5.

[109] S. Mittal, P. K. Das, V. Mulwad, A. Joshi, and T. Finin, "Cybertwitter: Using twitter to generate alerts for cybersecurity threats and vulnerabilities," in *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, IEEE, 2016, pp. 860–867.

[110] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pp. 529–540, 2007.

[111] NIST, *National vulnerability database*. [Online]. Available: https://nvd.nist.gov/.

[112] OffSec, "Exploit db dataset," 2024. [Online]. Available: https://gitlab.com/exploit-database/exploitdb/-/blob/main/files_exploits.csv?ref_type=heads.

[113] A. Okutan, P. Mell, M. Mirakhorli, I. Khokhlov, J. C. S. Santos, D. J. Gonzalez, and S. Simmons, "Empirical validation of automated vulnerability curation and characterization," *IEEE Transactions on Software Engineering*, 2023. DOI: 10.1109/TSE.2023.3250479. [Online]. Available: https://doi.org/10.1109/TSE.2023.3250479.

[114] A. Ozment and S. E. Schechter, "Milk or wine: Does software security improve with age?" In *USENIX Security Symposium*, vol. 15, 2006, pp. 93–104.

[115] S. P., "Securing cyberspace: Navigating zero-day vulnerabilities through discovery, disclosure strategies, and defence mechanisms," *International Journal of Computer Applications*, vol. 185, no. 22, pp. 12–19, Nov. 2023.

[116] R. Parla, "Efficacy of epss in high severity cves found in kev," *arXiv preprint*, 2024. DOI: 10.48550/arxiv.2411.02618.

[117] J. Pastor-Galindo, P. Nespoli, F. GÃşmez MÃąrmol, and G. MartÃ■nez PÃľrez, "The not yet exploited goldmine of osint: Opportunities, open challenges and future trends," *IEEE Access*, vol. 8, pp. 10 282–10 304, 2020. DOI: 10.1109/ACCESS.2020.2965257.

[118] E. Pauley, P. Barford, and P. McDaniel, "The cve wayback machine: Measuring coordinated disclosure from exploits against two years of zero-days," in *ACM Conference on Computer and Communications Security*, 2023, pp. 1–15. DOI: 10.1145/3618257.3624810.

[119] H. Perl, S. Dechand, M. Smith, D. Arp, F. Yamaguchi, K. Rieck, S. Fahl, and Y. Acar, "Vccfinder: Finding potential vulnerabilities in open-source projects to assist code audits," *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 426–437, 2015.

[120] F. Piessens, "A taxonomy of causes of software vulnerabilities in internet software," M. Vouk, Ed., pp. 47–52, 2002.

[121] V. Pothamsetty and B. A. Akyol, "A vulnerability taxonomy for network protocols: Corresponding engineering best practice countermeasures," in *Communications, Internet, and Information Technology 2004*, St. Thomas, US Virgin Islands, 2004.

[122] S. M. Rajasooriya, C. P. Tsokos, and P. K. Kaluarachchi, "Cyber security: Nonlinear stochastic models for predicting the exploitability," *Journal of Information Security*, vol. 7, no. 4, pp. 209–223, 2016. DOI: 10.4236/jis.2016.74017.

[123] S. M. Rajasooriya, C. P. Tsokos, and P. K. K. H. Kaluarachchilage, "Vulnerability life cycle exploitation timing modeling," *Journal of Information Security*, vol. 11, no. 2, pp. 133–150, May 2020.

[124] P. M. Rao, M. V. Nikhil, V. S. Manasali, T. S. Saketh, and S. Babu, "Relationship between factors affecting software vulnerabilities," in *2023 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, IEEE, 2023, pp. 181–185.

[125] C. Ravikumar, B. Naresh, P. Prasanna, N. Chander, E. A. Goud, and P. Prasad, "Exploring machine learning algorithms for robust cyber threat detection and classification: A comprehensive evaluation," 2024. DOI: 10.1109/iscs61804.2024.10581226.

[126] A. Rengarajan, "Emerging technological threats to cybersecurity," *International Journal of Innovative Research in Computer and Communication Engineering*, 2024. DOI: 10.15680/ijircce.2024.1205125.

[127] L. G. A. Rodriguez, J. S. Trazzi, V. Fossaluza, R. Campiolo, and D. M. Batista, “Analysis of vulnerability disclosure delays from the national vulnerability database,” in *Workshop de Segurança Cibernética em Dispositivos Conectados (WSCDC)*, SBC, 2018.

[128] S. Rosyada, F. A. Rafrastara, A. F. Ramadhani, W. Ghozi, and W. Yassin, “Enhancing xgboost performance in malware detection through chi-squared feature selection,” *Jurnal Sistem Informasi dan Komputer*, Nov. 2024. DOI: `10.32736/sisfokom.v13i3.2293`.

[129] M. Roytman and J. Jacobs, “The complexity of prioritising patching,” *Network Security*, vol. 2019, no. 7, pp. 12–16, Jul. 2019.

[130] J. Ruohonen, “A look at the time delays in cvss vulnerability scoring,” *Applied Computing and Informatics*, vol. 15, no. 2, pp. 129–135, 2019.

[131] C. Sabottke, O. Suciu, and T. Dumitras, “Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits,” in *24th USENIX Security Symposium*, 2015, pp. 1041–1056.

[132] R. C. Seacord and A. D. Householder, “A structured approach to classifying security vulnerabilities,” Carnegie Mellon University Software Engineering Institute, Tech. Rep. CMU/SEI-2005-TN-003, 2005.

[133] R. Sen and G. R. Heim, “Managing enterprise risks of technological systems: An exploratory empirical analysis of vulnerability characteristics as drivers of exploit publication,” *Decision Sciences*, vol. 47, no. 6, pp. 1073–1102, 2016. DOI: `10.1111/DECI.12212`.

[134] M. Shahzad, M. Z. Shafiq, and A. X. Liu, “Large scale characterization of software vulnerability life cycles,” *IEEE Transactions on Dependable and Secure Computing*, Jul. 2020.

[135] M. Shahzad, M. Z. Shafiq, and A. X. Liu, “A large scale exploratory analysis of software vulnerability life cycles,” in *2012 34th International Conference on Software Engineering (ICSE)*, IEEE, 2012, pp. 771–781.

[136] A. Sharma, S. Sabharwal, and S. Nagpal, “A hybrid scoring system for prioritization of software vulnerabilities,” *Computers & Security*, 2023. DOI: `10.1016/j.cose.2023.103256`.

[137] E. Shereen, D. Ristea, S. Vyas, S. McFadden, M. Dwyer, C. Hicks, and V. Mavroudis, “Sok: On closing the applicability gap in automated vulnerability detection,” 2024. DOI: `10.48550/arxiv.2412.11194`. [Online]. Available: `https://doi.org/10.48550/arxiv.2412.11194`.

[138] Z. Shi, N. Matyunin, K. Graffi, and D. Starobinski, “Uncovering cwe-cve-cpe relations with threat knowledge graphs,” *ACM Transactions on Privacy and Security*, vol. 27, pp. 1 –26, 2023. [Online]. Available: `https://api.semanticscholar.org/CorpusID:258427113`.

[139] Y. Shin and L. Williams, “An initial study on the use of execution complexity metrics as indicators of software vulnerabilities,” *Proceedings of the 7th International Workshop on Software Engineering for Secure Systems*, pp. 1–7, 2011.

[140] J. C. da Silva Santos, K. Tarrit, and M. Mirakhorli, “A catalog of security architecture weaknesses,” Apr. 2017, pp. 220–223. DOI: `10.1109/ICSAW.2017.25`.

[141] M. Souppaya and K. Scarfone, "Guide to enterprise patch management planning," National Institute of Standards and Technology, Special Publication 800-40r4, 2022. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-40r4.pdf.

[142] M.-O. Stehr and M. Kim, "Vulnerability clustering and other machine learning applications of semantic vulnerability embeddings," *arXiv.org*, vol. abs/2310.05935, 2023. DOI: 10.48550/arxiv.2310.05935. [Online]. Available: https://export.arxiv.org/pdf/2310.05935v1.pdf.

[143] O. Suciu, C. Nelson, Z. Lyu, T. Bao, and T. Dumitras, "Expected exploitability: Predicting the development of functional vulnerability exploits," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 377–394. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/suciu.

[144] H. Sun, G. Ou, Z. Zheng, L. Liao, H. Wang, and Y. Zhang, "Inconsistent measurement and incorrect detection of software names in security vulnerability reports," 2023. DOI: 10.1016/j.cose.2023.103477. [Online]. Available: https://doi.org/10.1016/j.cose.2023.103477.

[145] J. Sun, Z. Xing, X. Xia, Q. Lu, X. Xu, and L. Zhu, "Aspect-level information discrepancies across heterogeneous vulnerability reports: Severity, types and detection methods," *ACM Transactions on Software Engineering and Methodology*, 2023. DOI: 10.1145/3624734. [Online]. Available: https://doi.org/10.1145/3624734.

[146] J. Sun, Z. Xing, X. Xu, L. Zhu, and Q. Lu, "Heterogeneous vulnerability report traceability recovery by vulnerability aspect matching," in *IEEE International Conference on Software Maintenance and Evolution*, 2022. DOI: 10.1109/ICSME55016.2022.00024. [Online]. Available: https://doi.org/10.1109/ICSME55016.2022.00024.

[147] R. Syed and H. Zhong, "Cybersecurity vulnerability management: An ontology-based conceptual model," in *Americas Conference on Information Systems*, 2018.

[148] C. Tanga, M. Mulpuri, D. Mahalakshmi, P. K. Hemalatha, T. M. Pattewar, and N. Parveen, "Advanced deep learning techniques for information security vulnerability detection using machine learning," *Communications on Applied Nonlinear Analysis*, 2024. DOI: 10.52783/cana.v32.1862.

[149] O. Temizkan, R. L. Kumar, and S. Park, "Patch release behaviors of software vendors in response to vulnerabilities: An empirical analysis," *The Missouri Review*, Apr. 2012.

[150] Tenable, *What is vpr and how is it different from cvss?* https://www.tenable.com/blog/what-is-vpr-and-how-is-it-different-from-cvss, 2020.

[151] K. Thomas, F. Li, A. Zand, J. Barrett, J. Ranieri, L. Invernizzi, Y. Markov, O. Comanescu, V. Eranti, A. Moscicki, D. Margolis, V. Paxson, and E. Bursztein, "Data breaches, phishing, or malware? understanding the risks of stolen credentials," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA: Association for Computing Machinery, 2017, pp. 1421–1434. DOI: 10.1145/3133956.3134067.

[152] G. TOD-RĂĆILEANU, A.-M. DincĂČ, S.-D. Axinte, and I. C. Bacivarov, "Enhancing vulnerability management with artificial intelligence algorithms," *International Conference on Cybernetics and Informatics*, 2024. DOI: 10.19107/cybercon.2024.13.

[153] H.-T. Turtiainen and A. A. Costin, "Vulnberta: On automating cwe weakness assignment and improving the quality of cybersecurity cve vulnerabilities through ml/nlp," pp. 618–625, 2024. DOI: 10.1109/eurospw61312.2024.00075.

[154] M. Usman, "Securing the future: The role of neural networks and ai in advanced cyber defense mechanisms," *engrXiv*, 2024. DOI: 10.31224/4066.

[155] Ã. J. Varela-Vaca, D. Borrego, R. M. Gasca, and A. G. Marquez, "Feature models to boost the vulnerability management process," *Journal of Systems and Software*, vol. 193, p. 111 541, 2022. DOI: 10.1016/j.jss.2022.111541.

[156] D. T. Vasireddy, D. Dale, and Q. Li, "Cvss base score prediction using an optimized machine learning scheme," *IEEE Workshop on Security*, 2023. DOI: 10.1109/rws58133.2023.10284627.

[157] D. Walkowski, A. Kizza, and J. Vidal, "An analysis of commercial vulnerability management tools and their prioritization methods," 2021.

[158] L. Watkins and J. S. Hurley, "The next generation of scientific-based risk metrics: Measuring cyber maturity," *International Journal of Cyber Warfare and Terrorism*, vol. 6, no. 3, pp. 12–23, 2016. DOI: 10.4018/IJCWT.2016070104.

[159] S. Weber, P. A. Karger, and A. Paradkar, "A software flaw taxonomy: Aiming tools at security," in *Software Engineering for Secure Systems (SESS'05)*, 2005.

[160] E. Wheeler, *Threat and Vulnerability Management*. Syngress, 2011. DOI: 10.1016/B978-1-59749-615-5.00011-6.

[161] S. Woo, E.-J. Choi, and H. Lee, "A large-scale analysis of the effectiveness of publicly reported security patches," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 1, pp. 460–473, Jan. 2023.

[162] Q. Xiong, S. Lian, Z. Zeng, R. He, B. Zhu, and X. Yang, "An empirical analysis of vulnerability information disclosure impact on patch r&d of software vendors," *Journal of Intelligent & Fuzzy Systems*, vol. 44, no. 1, pp. 839–853, 2023.

[163] C. Yi-Wei, "Automated vulnerability scanning and event management in commercial solutions," 2020.

[164] M. K. Yogi, "A comprehensive study of zero-day attacks," *Journal of Information Technology and Digital World*, vol. 5, no. 3, pp. 45–58, Sep. 2023.

[165] C. S. Young, "Information technology risk measurements and metrics," *Information Security Science*, pp. 253–265, 2016. DOI: 10.1016/B978-0-12-809643-7.00012-7.

[166] A. A. Younis and Y. K. Malaiya, "Comparing and evaluating cvss base metrics and microsoft rating system," in *2015 IEEE International Conference on Software Quality, Reliability and Security*, IEEE, 2015, pp. 252–261.

[167] Y. Zheng, S. Pujar, B. Lewis, L. Buratti, E. Epstein, B. Yang, J. Laredo, A. Morari, and Z. Su, "D2a: A dataset built for ai-based vulnerability detection methods using differential analysis," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, IEEE, 2021, pp. 111–120. DOI: 10.1109/ICSE-SEIP52600.2021.00019.

[168] F. Zhou, H. Fan, Y. Liu, Y. Ye, H. Diao, Z. Wang, R. Rached, Y. Tu, and E. Davio, "Application of xgboost algorithm in rate of penetration prediction with accuracy," in *International Petroleum Technology Conference*, IPTC, 2022, D012S111R002. DOI: 10.2523/iptc-22100-ms.

[169] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[170] T. Zimmermann, N. Nagappan, and L. Williams, "Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista," *Proceedings of the Third International Conference on Software Testing, Verification and Validation*, pp. 421–428, 2010.