Universiteit Leiden

Master Computer Science

Exploring the landscape of CVE-related Proof-of-Concepts published on GitHub

Name:	Robin The
Student ID:	S2640309
Date:	01/02/2026
Specialisation:	Advanced Computing and Systems
1st supervisor:	Olga Gadyatskaya
2nd supervisor:	Yury Zhauniarovich

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Public proof-of-concept (PoC) exploit code plays an important role in the vulnerability ecosystem, supporting validation, development of defenses, and security research. At the same time, PoCs are dual-use artifacts and are widely disseminated through public code platforms such as GitHub. To better understand this landscape, we conduct a large-scale empirical study of CVE-referenced PoC repositories hosted on GitHub. We analyze multiple data snapshots collected throughout 2024, comprising more than 93,000 repositories and referencing 6,531 unique vulnerabilities referenced as CVEs. Our analysis examines repository activity, implementation characteristics, CVE coverage, and temporal relationships between vulnerability disclosure and repository creation, using metadata from the National Vulnerability Database (NVD). By combining historical snapshots with longitudinal monitoring, we capture both the static structure of the ecosystem and its evolution over time, including repository creation, deletion, and changes in availability. In addition, we apply social network analysis to repository ownership and forking relationships, introducing the notion of fork families to characterize code reuse and propagation. Our findings show that original PoC repository publication is concentrated among a small set of users. Whereas the majority of collected repositories are forks, indicating that reuse is more prevalent than independent PoC developments within the ecosystem. Repository creation clusters closely around vulnerability disclosure dates, and PoCs disproportionately target severe, remotely exploitable vulnerabilities. Social network analysis reveals a highly skewed structure with many small fork families and limited cross-year fork interaction. Overall, this study provides an empirical characterization of the structure and dynamics of the CVE-related PoC ecosystem on GitHub, offering insights into how exploit code is published, reused, and disseminated on public platforms.

Contents

1 Acknowledgements

I would like to express my sincere gratitude to everyone who has supported me throughout my academic journey, spanning both my bachelor's and master's studies.

First and foremost, I would like to thank Olga Gadyatskaya and Soufian El Yadmani for their extensive guidance, mentorship, and continued support during my bachelor's thesis, master's thesis, and the research project that, despite an initially difficult process, ultimately resulted in a publication at *19th USENIX WOOT Conference on Offensive Technologies*! Their commitment, patience, and feedback were essential, and I am particularly grateful for the trust they placed in me that shaped both this thesis and my broader research interests, and I look forward to continuing our collaboration beyond this thesis. I would also like to thank Yury Zhauniarovich for acting as a second supervisor and for his encouragement and thoughtful feedback, particularly in relation to the broader research this thesis was part of. I would like to especially thank Soufian El Yadmani for the knowledge, support, and inspiration he provided not only in an academic context, but also in professional and non-academic settings. I am grateful for the opportunity to continue collaborating in a professional setting.

My sincere thanks further go to my fellow students, and also friends, in particular Roos Wensveen and Renée Gerritse, for their support, collaboration, and the enjoyable work we shared during our studies. Their enthusiasm and teamwork made this journey both more productive and more enjoyable. Finally, I would like to thank my parents for their endless patience, encouragement, and support throughout this bit-longer-than-expected academic journey. Their understanding made it possible for me to grow professionally, start my career, and continue to develop both academically and personally. Ultimately, I hope that through my work, whether directly in practice or indirectly through research, I can help make our rapidly changing, computer-world a little safer.

2 Introduction

Publicly disclosed vulnerabilities play a central role in the field of cybersecurity and research. The Common Vulnerabilities and Exposures (CVE) system provides standardized identifiers for known vulnerabilities, enabling coordination between security researchers, vendors, and practitioners. In addition, it serves as the foundation for vulnerability databases such as the National Vulnerability Database (NVD). Alongside formal disclosure, technical artifacts, such as proof-of-concept (PoC) and exploit code have become increasingly important in the vulnerability ecosystem. PoCs are purpose-built scripts or programs that illustrate how a vulnerability can be exploited and are widely used for validation and mitigation. GitHub has emerged as a prominent platform for the hosting and disseminating of such artifacts due to its ease of use and collaborative features. As a result, large collections of CVE-referenced PoCs are publicly available, evolving continuously as repositories are created, updated, forked and shared.

However, this wide availability introduces a dual-use dilemma. Although public PoCs contribute to transparency and preparedness, they can also lower the difficulty of exploitation for adversaries. Especially when exploits are shared before or shortly after vulnerability disclosure or before patches are available. Understanding the dynamics of PoC publication, dissemination, and reuse is therefore critical for understanding the defensive value and potential risk. Prior work has provided valuable insights into vulnerability disclosure and exploit development. While work examining CVE-linked PoCs as a collaborative ecosystem on GitHub is still limited.

In this work, we address this by conducting a repository-centric analysis of CVE-related PoCs hosted on GitHub. Rather than treating repositories as single artifacts, we model them as part of a dynamic and collaborative ecosystem that evolves over time. The foundation of our approach is extensive data collection, comprising multiple snapshots of GitHub repositories associated with CVEs. All collected and stored using a custom-built monitoring and storage system designed to capture changes over time. This enables us to observe not only the presence of PoCs, but also their publication, reuse, and social structure. Building on this data, we perform systematic cleaning and de-duplication to uniquely identify repositories across snapshots. We analyze the programming languages used, the distribution of CVEs, and the temporal relationship between CVE disclosure and repository publication. To capture the collaborative nature of the ecosystem, we further construct network representations of repository ownership and fork relationships. In particular, we introduce the notion of *fork families*, defined as connected components in the fork graph, which represent groups of repositories derived from a common PoC-repository lineage.

This network-based perspective enables an alternative view on PoC activity, adding to the analysis of raw repository counts. By examining the size, ownership, and labeling of fork families, we provide additional insight into how PoCs disseminate over time from a higher-level perspective.

The main contributions of this paper are as follows.

(i) the construction of a large-scale dataset of CVE-related repositories on GitHub, collected through multiple snapshots and supported by an additional custom monitoring system, providing foundation for studying the distribution and dynamics of public PoCs at scale.

(ii) a systematic analysis of PoC characteristics, including programming language usage, CVE coverage, and publication timing, revealing the coverage across vulnerabilities and distinct patterns in how and when PoCs are published on GitHub.

(iii) a longitudinal monitoring system of PoC availability, capturing creation and deletions events over time, which we use to track the dynamic availability of PoCs by capturing additions and deletions over time.

(iv) a network-based modeling of PoC collaboration through ownership and fork graphs, including the introduction of fork families as a unit of analysis, revealing patterns in ownership and forking across all collected repositories.

(v) a temporal analysis of fork-family activity, providing an additional perspective on the evolution of exploit-related communities over time.

This research is organized as follows. Section 5 describes the data collection methodology, storage design, monitoring system, cleaning process, and metadata extraction and enrichment. Section 6 presents the analysis of repositories, CVEs, and publication timing, then reports insights from the longitudinal monitoring system on repository creation and disappearance. Finally introduces the network models and fork-family construction, followed by analysis of fork-family activity. Section 7 presents the discussion of the analysis and the limitations of the research. Finally, Section 8 summarizes the findings and future work.

3 Preliminaries and Related Work

This section introduces key concepts underlying this research, including Common Vulnerabilities and Exposures (CVE), which serve to identify, define, and catalog known security vulnerabilities, along with exploit code that demonstrates how these vulnerabilities are leveraged in attacks. Additionally, an overview of GitHub is provided, with its role as a collaborative platform for sharing exploit code. Collectively, these concepts form the foundation for a deeper understanding of the study. Moreover, this section reviews related work, summarizing the current state of knowledge on proof-of-concepts for CVEs and GitHub repositories, and identifying gaps in the literature where no prior studies have yet combined these two areas.

3.1 Common Vulnerabilities and Exposures (CVE)

The Common Vulnerabilities and Exposures system serves as a central repository for publicly disclosed information-security vulnerabilities and exposures in widely used systems. This system serves as a reference standard, enabling clear and consistent identification of security vulnerabilities. Each record in the CVE database receives a unique CVE ID, consisting of a year identifier and a sequence of digits that uniquely identify the vulnerability for that year (e.g., CVE-2021-42288). All CVE records include a description of the vulnerability, the current state of the record (reserved, published, or rejected), the record's creation date, and relevant references. The publishing of these CVE IDs is done by CVE Numbering Authorities (CNAs), under supervision of the central MITRE Corporation[1]. Each published CVE entry goes through stages designed to validate security vulnerabilities. Initially, a CVE entry is marked as "reserved" when assigned by a CVE Numbering Authority (CNA). Following the verification process, it progresses to "published" status, meaning the vulnerability has been validated and is now publicly accessible with a standardized description and metadata. If a CVE later is proven to be invalid, the entry is marked as "rejected" to prevent the spread of incorrect information.

3.2 National Vulnerability Database (NVD)

The National Vulnerability Database[2] is a U.S. government repository that builds upon the CVE system by providing additional metadata and context for each identified vulnerability.
It incorporates information such as severity scores based on the Common Vulnerability Scoring System (CVSS) [3], which help prioritize vulnerabilities based on their potential impact. The CVSS system provides a way to capture the principal characteristics of a vulnerability and produce a numerical score reflecting its severity. The numerical score can then be translated into a qualitative representation (such as low, medium, high, and critical) to help organizations properly assess and prioritize their vulnerability management processes [SIG24]. In addition to severity, CVSS also includes exploit characteristics, such as the *Attack Vector* metric, which distinguishes whether exploitation is possible over the network or requires local (or other) forms of access. The NVD also include a variety of other data, such as affected software versions, vendor-specific patches or advisories, and mitigation recommendations. This enriched data repository allows security professionals to gain a deeper understanding of CVEs.

[1] MITRE corporation (`https://www.mitre.org/`)
[2] NVD (`https://nvd.nist.gov/`)
[3] `https://www.first.org/cvss/specification-document`

Although the CVE system offers a standardized method for identifying and cataloging vulnerabilities, and the NVD enhances this by providing additional context, there remains a critical gap in linking these vulnerabilities to active exploits in the wild. Understanding how specific CVEs are being actively exploited in real-world attacks is essential for effectively prioritizing and addressing security vulnerabilities.

3.3 Exploit and Proof-of-Concept Code

In the context of Common Vulnerabilities and Exposures (CVEs), exploit code refers to purpose-built scripts, binaries, or programs that adversaries can use to leverage a vulnerability to achieve impact on a target system such as remote code execution, privilege escalation, or denial of service. Once a CVE is published for a known vulnerability, researchers or security professionals often develop a proof-of-concept (PoC), typically a minimal and controlled demonstration that shows the vulnerability is not just theoretical but can be exploited in real-world scenarios. A PoC is typically designed to confirm the existence and potential impact of the vulnerability without causing real harm, helping to understand the risk and urgence of a vulnerability. PoCs can be released publicly or shared within closed security communities, encouraging vendors, companies and organizations to respond to security vulnerabilities. However, when exploit code is released into the wild, it often evolves beyond a PoC into a fully functional tool that hackers can use to target and compromise live systems. This highlights the dual nature of publicly available PoCs and exploit code, serving both as a diagnostic tool and, when misused, as a weapon for malicious actions.

During penetration testing or security assessments, information security professionals aim to identify known vulnerabilities to ensure that they are patched and mitigated. However, it is not sufficient to identify a system as vulnerable, penetration testers must also demonstrate that the vulnerability can be exploited. Professional frameworks like Metasploit [Rap24] and trusted databases such as Exploit-DB [Sec24] offer exploits for numerous known weaknesses, which are also cataloged in the CVE database. However, not all CVEs have exploits available in these sources. To find potential proof-of-concept exploits, beyond these platforms, penetration testers often turn to public code repositories, such as GitHub[4].

3.4 GitHub

GitHub is a cloud-based platform designed for storing, sharing, and collaborating on code. The code is stored in a **repository** on GitHub, allowing developers to showcase or share their work [Git24]. Activity within these repositories includes code commits, which represent discrete, version-controlled changes to the repository, issues, which are structured discussions used for documentations, and pull requests which facilitate integration of code modifications. Additionally, GitHub provides several interactive features to engage with repositories. The **fork** action enables users to create their own personal copy of a repository, allowing them to experiment or contribute without affecting the original project. The **star** action enables users to mark a repository as a favorite. The **watch** action allows users to keep track of all activity within a repository, including commits, issues, and pull requests. Together, these actions serve as metrics for gauging the popularity and potential utility of specific repositories on GitHub, including those containing PoCs.

[4] GitHub (`https://github.com/`)

Sources such as Exploit-DB validate the effectiveness and legitimacy of PoCs, public code sharing platforms, such as GitHub, lack such exploit vetting process. The absence of this process, means that PoCs found may be unreliable or even contain malicious content, posing a threat to anyone running them. GitHub's ease in allowing code to be published may result in less thorough review of code, making it a very attractive but potentially dangerous platform for CVE exploit sharing.

3.5 Social Network

Social networks are systems in which a set of entities (nodes) are connected by relationships or interactions (edges), such as communication, information sharing, or collaboration. Social Network Analysis (SNA) studies the structure of these systems providing a view of relations within a network. In this context, GitHub and its repositories exhibit properties of a social network: developers interact through ownership, contributions, and forks, forming a graph of relationships rather than isolated artifacts. This perspective enables the identification of patterns that are not visible through simple counts or metadata. As such, SNA offers a useful analysis for understanding how the exploit-related ecosystems behaves as a social network.

In addition to platforms like GitHub, exploit code is discussed and shared on social media platforms such as X (formerly known as Twitter)[5] and Reddit[6]. These platforms serve as a space where security researchers, hackers and enthusiasts actively engage in sharing and discussing proof-of-concept exploits for known vulnerabilities, including technical details and discussing the impact of these vulnerabilities. The open and accessible nature of these platforms plays a critical role in how security-related knowledge is spread in the real world.

Although these platforms contribute significantly to the spread of security knowledge, this highlights the need for research into how PoCs are shared and presented outside strictly controlled environments. This research specifically focuses on GitHub as a central hub for sharing PoC exploits. By examining GitHub, the research aims to provide critical insights into how PoCs are shared and their impact on cybersecurity practices.

3.6 Related Work

The use of public code repositories, such as GitHub, for sharing and collaborating on exploit code has significantly contributed to the widespread availability of proof-of-concept exploits for vulnerabilities cataloged in the CVE database. GitHub serves as an open platform where security researchers, developers, and even malicious actors can contribute code that can demonstrate exploitation. While GitHub's openness facilitates knowledge sharing, the increasing trend of publishing PoCs on the platform raises concerns about the role of GitHub as a source for exploit code. This section provides a review of related work on the Common Vulnerabilities and Exposures system, exploit code, GitHub and it's role in the publication of exploit code, specifically through proof-of-concept exploits.

[5] X(Twitter) (`https://x.com/`)

[6] Reddit (`https://www.reddit.com/`)

3.6.1 CVE exploits

Studying **Common Vulnerabilities and Exposures** (CVEs) plays a crucial role in understanding the evolution of vulnerabilities. Research in this area explores the CVE system itself, the impact of CVEs and the dynamics of the lifecycle of CVEs. The study by [MYB19] examines vulnerability reports in the CVE database, analyzing the descriptions of CVEs from 1999 to 2019. The research performs mapping techniques to align CVEs to vulnerability categories defined by the OWASP Top 10, revealing trends of certain vulnerability types. This helps understanding what vulnerability types have been prevalent and what have been diminished. Building upon the evolution of CVEs, the work in [PFNS23] investigates the lifetime of CVEs, highlighting the dynamics of how vulnerabilities are handled over time. Another important aspect of research about CVEs is the prediction of severity and exploitability of vulnerabilities. Public vulnerability databases such as the National Vulnerability Database use the **Common Vulnerability Scoring System** to assign various scores to CVEs. The study in [OM22] approaches the automation of predicting CVE severity and exploitability, utilizing CVE descriptions from the NVD within a Convolutional Neural Networks (CNN).

Moreover, understanding the relationship between vulnerabilities and their fixes is crucial because it shows how vulnerabilities are addressed in practice. The research in [BNM21] focuses on collecting CVEs and their corresponding fixes from open-source projects, aiming to uncover insights into the dynamics of vulnerabilities. By linking CVEs directly to their fixes, this work provides a foundation for more precise analysis of how vulnerabilities are addressed. The study by [LCB22], surveys data-driven approaches to Software Vulnerability (SV) assessment and prioritization. It presents an overview of past research, outlines best practices, and proposes solutions to current limitations in SV mitigation.

On a larger scale, [NZ10] provides an analysis of the CVE database, categorizing vulnerabilities to identify patterns and trends. Regardless of being an older study from 2010, it remains relevant for tracking the popularity of different types of vulnerabilities over time. More recent work by [DSG21] provides an investigation of severity of CVEs. It analyzes CVSS metrics and classifies CVEs accordingly, aiming to explore how the number of vulnerabilities evolves over time across different severity levels.

Research on **exploit-code** focuses on understanding how vulnerabilities, cataloged in the CVE database, are weaponized by attackers. A recent study of exploit code by [FNFV23], examines the relationship between source code and exploits, proposing semantic analysis to enhance the detection and prioritization of vulnerabilities. In the study by [SNL+22], the authors introduce Expected Exploitability (EE), a metric to predict the likelihood of exploitation over time. It compares against assessments like CVSS scoring by adopting a dynamic, time-sensitive approach and leveraging artifacts published after vulnerability disclosure, such as technical write-ups and proof-of-concept exploits.

Building on this research, several recent studies have examined publicly available vulnerability PoC reports. The work introducing POSVIA [DWCH25] analyzes inconsistencies in affected software version information across multiple PoC sources, showing that discrepancies are common and proposing automated detection using deep learning. Complementary research [WDZ+25] investigates information completeness in PoC reports, demonstrating that

publicly shared PoCs frequently lack essential details and proposing multi-source fusion techniques to complete missing information. A broader large-scale measurement study of PoCs in the wild [DLC+25] further highlights systemic limitations, finding that most CVEs lack publicly available PoCs and that existing PoC reports often miss substantial portions of required components for understanding and reproduction. Together, these studies emphasize that while PoCs are widely used for vulnerability validation, their availability, consistency, and completeness remain uneven, motivating further systematic analysis of PoC ecosystem.

In our previous research [YTG22, The22, YTG25], we focused on uncovering the presence of **malicious repositories** associated with CVE (Common Vulnerabilities and Exposures) exploits on GitHub. In the previous study we identified several indicators that could signal the presence of malicious contents in repositories. By analyzing these indicators, we provided a framework for detecting potentially harmful repositories and underline the importance of repository analysis. The study also analyzed when repositories appeared over time and their basic characteristics, including programming languages. In this paper, we extend the scope to analyze GitHub repositories in a broader sense, aiming to gain insights into CVE-related repository characteristics, trends, and their relationships to CVE exploits. This broader perspective enables a deeper understanding of the GitHub ecosystem by specifically analyzing repositories associated with CVE exploits.

3.6.2 GitHub repositories

GitHub has become a prominent data source for research across various domains, offering insights into developer activity, software popularity, and collaborative behavior. Many studies have leveraged GitHub data to analyze different aspects of the platform. For example, prior research examined challenges regarding the collection of GitHub data and the processing of it. [MV18] provides a comparison of three methods for gathering GitHub data: GitHub REST API, GitHub Archive, and GHTorrent. The study highlights their behavior, available data, and use cases, recommending the GitHub REST API for retrieving the most recent and up-to-date data, hence we used the **GitHub API** in our own research to collect relevant data.

Another study [CS15] explores GitHub user behavior and project success on 100,000 projects and 10,000 users collected via the GitHub API. The analysis uses statistical methods, resulting in seven success rules for projects, providing valuable insights into the GitHub ecosystem. Additionally, a study [DYZ+20] on promotion services within GitHub identified accounts manipulating the platform's reputation system, such as through paid star and fork services. The study uncovered over 60,000 suspected promotion accounts, which may financially benefit from promoting on GitHub.

More recently, He et al. [HYB+24] conducted a systematic, global, and longitudinal measurement study of fake stars on GitHub using *StarScout*, a detector for anomalous starring behavior over GitHub metadata. They show that fake-star activity surged in 2024 and that campaigns frequently promote short-lived malicious repositories hosting phishing malware. These results highlight that GitHub's popularity signals can be manipulated at a large scale. The study [KGB+14], conducted an exploratory analysis of GitHub repositories, investigating user interactions. The findings emphasized that most repositories are personal and inactive, highlighting the importance of addressing data properly.

Several studies have focused on identifying factors influencing repository popularity. One study [BHV16] analyzed the popularity of repositories, measured by the number of stars, and linked it to factors such as programming language. The research made use of the GitHub API and encountered certain limitations, which we have been valuable in creating the methodology in our study. Another study [ZZM14], investigated the use of folders in over 140,000 GitHub projects, analyzing the relationship between folder use and project popularity, measured by the numbers of forks. The research found that standard folder names such as `document`, `testing`, and `examples` are associated with an increased chance of repositories being forked.

Others studies have examined forking on GitHub as a mechanism for collaboration and project evolution. [JLH+17] researched why and how developers fork repositories on GitHub, combining large-scale data analysis with survey responses to study motivations. Their findings indicate that forking is often used to contribute to original projects, for example via pull requests, and that forking behavior is influenced by language preferences and owner attractiveness. Related work around forks, has focused on characterizing collaboration across multiple repositories through forking. [BB14] introduced metrics aimed at analyzing multi-repository code bases, by classification of commits and modeling interactions across repositories, ultimately capturing collaboration patterns that emerge when development is distributed over multiple forks.

Complementary to these analyses around forks, more recent work has explored fork ecosystems from a visualization perspective. The study [CCSK24] investigates how developers and project managers perceive and explore complex fork ecosystem. Their results highlight the complexity of large fork ecosystems and the challenges faced when reasoning about relationships among many related repositories.

Given that this research focuses on analyzing repositories related to CVEs, these findings highlight the importance of carefully considering the quality of the data and potential biases when studying GitHub repositories. The challenges in the studies indicate the need for a sophisticated approach when analyzing GitHub repositories. Additionally, understanding the factors that influence the popularity and structure of repositories is key to the analysis of repositories in this research. To the best of my knowledge, this has not been extensively explored in previous research focused on CVE-related GitHub repositories.

3.6.3 Social network data

The analysis of social network data has become increasingly important in the context of cybersecurity and exploit detection. Data from social platforms can be leveraged to identify patterns, detect emerging threats, and predict exploitation of vulnerabilities. Research by [HBL+19] investigates user-generated content related to security vulnerabilities across three digital platforms: Reddit, Twitter, and GitHub. The study shows that activity on these social media platforms can serve as an indicator for popularity on GitHub. Another study by [SSD15] focuses on detecting the sharing of information related to security vulnerability exploitation on Twitter. They developed techniques to pinpoint relevant discussions on the platform and extract meaningful insights regarding CVE exploitation. Study [SCG19] has demonstrated that GitHub is a valuable source of information regarding potential attack vectors. Similarly, in [SSM+20], they found that CVE-related information is shared on GitHub even before a vul-

nerability is officially disclosed.

The research [TBLJ13] performs network analysis on GitHub, examining the relationships between 100,000 GitHub repositories and 30,000 creators of these repositories. They identified influential developers, projects and relations within the platform's social network. Research [LRM14] provides a quantitative analysis of GitHub, characterizing it as both a social network and a collaborative platform. The study examines over 183 million events across 2.19 million users and 5.68 million repositories, revealing patterns in GitHub interactions. Additionally, research [YYWW14] examines the rapid growth of GitHub, comparing its user growth to traditional open-source communities and identifying social behavior patterns within the platform, such as group, star, and hub structures by analyzing relations within the network.

Prior work has applied network-based analyses to GitHub in order to understand its social and collaborative structure. The research by [TBLJ13] performs large-scale network analysis of GitHub, examining relationships between over 100,000 repositories and 30,000 repository creators, and identifying influential developers, projects, and interaction patterns within the platform's social network. Similarly, [LRM14] provides a quantitative characterization of GitHub as both a social network and a collaborative platform, analyzing more than 183 million events across 2.19 million users and 5.68 million repositories, revealing patterns in GitHub interactions. Additionally, [YYWW14] investigates GitHub's rapid growth relative to traditional open-source communities and identifying social behavior patterns within the platform, such as group, star, and hub structures by analyzing relations within the network.

More studies have focused on the influence and centrality within GitHub's social network. For example, [HWRC18] analyzed user influence by constructing a GitHub developer social network and proposing a Following–Star–Fork–Activity-based approach to measure user influence. Their study evaluates influence along multiple dimensions, including popularity, centrality, content value, contribution, and activity, and combines these measures to identify influential users. Together, these works demonstrate how network and interaction data from GitHub can be used to study social structure, influence, and collaboration.

Building on these findings, this research incorporates social network analysis to uncover insights into the collaborative behavior between repositories related to CVEs. This analysis will further help understanding the interactions and relationships between users and repositories in the context of CVE exploits.

These studies collectively contribute to the understanding of various aspects related to CVE exploit code, repository structures, and social interactions within platforms like GitHub. Although these studies offer valuable insights, none have specifically focused on analyzing repositories containing proof-of-concept exploits for CVEs and assessing them as a landscape on GitHub. This research fills this gap by focusing on the specific subset of repositories related to CVE exploits presented on GitHub. We extend this work by incorporating social network analysis to explore how exploit code is disseminated within the GitHub ecosystem. This approach aims to enhance our understanding of how vulnerabilities are openly presented on the GitHub platform.

4 Research Design

This section outlines the research aims, objectives, and questions that guide the exploration of proof-of-concept exploits for vulnerabilities listed in the Common Vulnerabilities and Exposures database, particularly focusing on their presence in GitHub repositories. Next the methodology for exploring these aims and questions, including analyzing repository metrics and exploit code dynamics, is also discussed in this section. The primary research question of this research is to investigate the characteristics, popularity, and dynamics of CVE exploit code published on GitHub.

4.1 Research Aims and Objectives

The research pursues the following aims and objectives:

- **Characterize the CVE–PoC repository ecosystem on GitHub.** Quantify the composition of the collected data, including the number of repositories, the split between original and forked repositories, and the number of unique repositories identified through the proposed hashing and deduplication mechanism. In addition, analyze dominant programming languages to understand how PoC code is typically implemented. *Research question: What is the size and structure of the CVE-related PoC ecosystem, how much repository content is duplicated via forks or potential re-uploads, and which implementation languages are the most common?*

- **Study vulnerability targeting and metadata.** Analyze which CVEs are most frequently referenced and whether CVE targeting differs between original and forked repositories. Characterize referenced vulnerability metadata such as CVSS score and attack vector. *Research question: Which vulnerabilities are most commonly targeted by PoCs on GitHub, and which vulnerabilities receive the highest community engagement as reflected by forks?*

- **Measure temporal dynamics of PoC publication.** Compare repository creation dates with CVE disclosure dates to assess how quickly PoCs appear, and to identify cases where repositories were created before official vulnerability publication. Using custom monitoring to quantify daily additions and deletions of repositories. *Research question: How does PoC publication timing relate to CVE disclosure, and what does daily monitoring reveal about repository creation and deletion?*

- **Model ownership and forking using social network analysis.** Construct and analyze ownership and fork graphs to assess the concentration of activity, such as highly active users and forking behavior. Introduce *fork families* as connected components within the fork graph to study fork-based communities. *Research question: How concentrated is PoC activity among users, and what does the fork network reveal about forking communities?*

4.2 Methodology

This study employs a combination of both quantitative and qualitative methods to research CVE-related GitHub repositories, with the primary aim to investigate the characteristics, popularity, and dynamics of CVE exploit code published on GitHub.

Data. This research utilizes a set of comprehensive snapshots of GitHub repositories referencing vulnerabilities recorded in the Common Vulnerabilities and Exposures database. The snapshots were created by systematically searching GitHub, using the GitHub API, for repositories referencing CVE identifiers (e.g., CVE-YYYY-XXXX), tagged with vulnerabilities from 2016 to 2024. During the process, both the repository content and associated metadata were collected, providing a historical baseline of available PoC code for security vulnerabilities and enabling for analysis of exploit code.

To capture ongoing trends and repository updates, a real-time monitoring system was implemented to track new and updated repositories on a daily basis. This system continuously queries GitHub's API to identify new or modified repositories linked to CVEs from 2018 to 2024, gathering only metadata to ensure efficient tracking and analysis. Key metadata collected includes repository creation and update dates, number of stars and forks, indicating popularity and engagement, and contributor information. This monitoring system allows for analysis of repository growth, evolution of exploit code over time, and the dynamics of GitHub engagement around security vulnerabilities. By combining a historical snapshot with ongoing monitoring, this dual approach facilitates a detailed investigation into the lifecycle and characteristics of CVE-related exploit code.

Analysis overview. Using the combined snapshots and monitoring system, we first report statistics on the collected repositories. We then analyze characteristics such as the dominant programming languages reported for the repositories. To capture engagement and duplication, repositories are labeled as original or forked repositories and further consolidated using a hashing-based deduplication technique to report the number of unique repositories.

Next, we study what vulnerabilities are targeted by extracting referenced CVE identifiers and analyzing the most frequently targeted CVEs, as well as for both original and forked repositories. To assess whether these distributions are different, we apply a statistical comparison between the original and fork subsets. Additionally we enrich referenced CVEs with vulnerability metadata to characterize the types of vulnerabilities targeted in repositories.

To analyze timing, we compare CVE disclosure dates with the repository creation dates and examine the resulting distributions, while also focusing on cases where repositories are created before official CVE publication. As well as monitoring the daily additions and removals of repositories to provide insight. Finally, we model collaboration and forking through social network analysis by constructing an ownership graph and fork graph. From the fork graph, we derive *fork families* as connected components to capture fork-based communities, and we characterize these groups by size, owner diversity, and temporal CVE-year label.

5 Data Collection

5.1 Overview and Objectives

This chapter outlines the processes and methodologies used to collect data on GitHub repositories containing proof-of-concept exploits associated with vulnerabilities listed in the Common Vulnerabilities and Exposures database. The aim is to comprehensively gather and organize information on repositories that reference CVE identifiers from 2016 to 2024, creating a dataset that includes both historical data and real-time updates. This dataset will provide the foundation for analyzing the trends, characteristics, and community dynamics of public PoC exploit code on GitHub.

The data collection process consists of two main components:

1. **Snapshot Collection:** Initial creation of a historical dataset of repositories containing PoC exploit code for CVEs from 2016 to 2024.
2. **Real-Time Monitoring:** Ongoing tracking of GitHub for new or updated repositories referencing CVE identifiers to maintain an up-to-date view of the current PoC landscape.

Objectives:

- **Identify relevant repositories:** Systematically identify and collect all GitHub repositories that include references to CVE identifiers from 2016 to 2024, focusing on PoC exploit code and other related data.
- **Capture (meta)data:** Downloading each repository and gathering detailed metadata on each repository, including creation date, update history, star and fork counts, contributor information, descriptions and CVE metadata, facilitating a comprehensive understanding of the repository's characteristics.
- **Establish real-time monitoring:** Develop a monitoring system that queries GitHub daily to track new or modified repositories referencing CVE identifiers. By focusing on the dynamics rather than content, the system provides insight into repository growth, evolution of exploit code over time, and the dynamics of GitHub engagement around security vulnerabilities
- **Enable (longitudinal) analysis:** By combining historical data with real-time monitoring, a comprehensive dataset is created to support extensive analysis, revealing trends in the creation, popularity, and evolution of PoC exploit code on GitHub. The historical dataset provides a strong foundation, offering an in-depth view of existing repositories, while the real-time monitoring system focuses on capturing dynamics. This approach allows for the analysis of both past and current developments, offering valuable insights into how PoC exploit code is presented and evolves over time.

This chapter will present the technical details and reasoning behind each step of the data collection process, establishing a reliable dataset for further exploration and analysis of GitHub-hosted PoC exploit code associated with CVEs.

5.2 Technical Implementation

This section outlines the technical methods used to collect, process, and analyze repositories related to publicly disclosed Common Vulnerabilities and Exposures. The implementation includes search term optimization, data gathering, CVE-ID extraction, and filtering techniques applied to metadata and content fields. Together, these techniques enabled the identification of relevant GitHub repositories for further analysis.

5.2.1 Search term optimization

To identify GitHub repositories containing references to CVE identifiers from 2016 to 2024, a systematic approach was applied. The goal was to focus on repositories hosting proof-of-concept exploit code. This process required a series of exploratory queries and iterative search term refinements to optimize the identification of relevant repositories.

The methodology started by testing variations of key terms such as "CVE," "PoC," "exploit," and specific CVE year patterns. This included targeting formats like CVE-2016-XXXX through CVE-2024-XXXX to ensure coverage of repositories containing exploit code for all CVE-years. The CVE-YEAR identifier was chosen as the most efficient method, as it provided pre-grouped repositories by CVE-year, simplifying the search process and improving the focus on repositories with relevant data.

The use of the search term CVE-YEAR was analyzed to ensure its effectiveness in retrieving relevant repositories. A key focus was examining the influence of letter casing and dash variations on query results.

Case sensitivity. Initial testing explored both lowercase (cve-YYYY) and uppercase (CVE-YYYY) variations. It was observed that GitHub's search functionality is case-insensitive in these cases, meaning that results remain consistent regardless of whether uppercase or lowercase queries are used. This insight simplified the design of search queries by removing the need for case-specific searches.

Dash variations. A more detailed investigation examined the use of different dash types within CVE-YEAR. Variations included the **standard hyphen**, **minus sign**, **en dash**, and **em dash**. For the CVE year 2022, specific queries were conducted using each of these dash types to evaluate their effectiveness. The results are presented in table 1, summarizing the number of repositories retrieved by each query. As shown in this table, the repository counts vary depending on the dash type used in the CVE query. The hyphen produces a higher number of results compared to other dash types, suggesting it is the most effective symbol for retrieving relevent CVE-related repositories.

Although the number of results obtained through different dash types provides an initial understanding of the query performance, this alone is not sufficient to determine which dash type is most effective in capturing all relevant repositories. Therefore, all repositories returned by the queries have been downloaded and identified for further examination.

Table 1: Repository counts for different dash types (CVE-Year: 2022).

Kind	Character	Unicode	CVE-2022 (Forks Excluded)	CVE-2022 (Forks Included)
Hyphen	-	U+002D	1901	11405
Minus Sign	−	U+2212	1005	7831
En Dash	–	U+2013	1005	7831
Em Dash	—	U+2014	1005	7831

The repositories retrieved by dash types: minus sign, en dash, and em dash, are fully captured by the hyphen's results. A total of 896 repositories were captured by the hyphen that were not retrieved by any of the other dash variants. This comparison highlights that the hyphen (-) serves as a superset over all other dash variants, including the minus sign (−), en dash (–), and em dash (—). This conclusion aligns with the table's findings, where the hyphen resulted in a larger number of repositories in both fork-included and fork-excluded queries to GitHub. This analysis was conducted to ensure greater accuracy by verifying that no repositories were missed when querying GitHub using the hyphen, thus maximizing coverage and simplifying the required queries. This investigation highlights the importance of refining search term structures to achieve comprehensive and reliable results.

The final optimized queries yielded a robust dataset of repositories containing PoC code for CVEs for the specified years. This dataset provides a valuable resource for security researchers, enabling deeper analysis of exploit patterns, attack vectors, and trends in vulnerability exploit sharing.

5.2.2 Data collection pipeline setup

The data collection process utilized the GitHub API to systematically retrieve repositories referencing CVE identifiers from 2016 to 2024. The API endpoint used for querying is:

`https://api.github.com/search/repositories?q={}+fork:true+is:public`

The pipeline was implemented using Python to query the GitHub API. The steps in the pipeline are as follows:

1. Query the GitHub API for repositories referencing `CVE-YEAR`, ensuring public repositories are included as well as forks.
2. Retrieve all results for the initial query, iterating over paginated results.
3. For each repository identified:
   (a) Query the API to fetch detailed user data of the repository owner.
   (b) Query the API to fetch detailed repository metadata.
   (c) Clone the repository locally for further analysis.

The pipeline was configured to collect an (initial) snapshot of repositories by querying all relevant CVE identifiers. This process captured essential metadata, including the description, creation date, last update date, and activity metrics such as stars and forks.
This systematic approach ensures the comprehensive collection of repositories with metadata relevant to CVE identifiers.

5.2.3 Snapshot

Initial snapshot. The first snapshot of repositories was collected by querying CVE identifiers for the years 2016 to 2024. This data collection occurred over the period, between **March 30, 2024, and April 8, 2024**. The process took multiple days due to rate limiting by the GitHub API, which restricts the number of queries that can be made within a specific timeframe. Despite these constraints, repositories were systematically retrieved to ensure comprehensive coverage of public repositories referencing CVE identifiers within this range.

Secondary snapshot. In addition to the initial snapshot, additional data collection was conducted to capture more relevant and up-to-date repositories for CVE year 2024. This process took place on **August 28, 2024**. The focus on 2024 was driven by the natural dynamics of the vulnerability landscape, where the most recent vulnerabilities tend to be shared, updated, and discussed more actively compared to older ones. This supplemental collection ensured that newer repositories referencing CVE identifiers from 2024 were included, enhancing the dataset's relevance and completeness.

5.2.4 Data Storage

The collected GitHub repositories are stored in a structured format to facilitate efficient data management and analysis. The stored structure ensures that all relevant repository data and metadata are systematically archived and timestamped.

Directory structure. Repositories are stored within a top-level directory categorized by the CVE year, allowing for logical grouping and easier navigation. Each repository is stored under a directory named using the format `<userName_repoName>`, which uniquely identifies the repository by combining the GitHub username and repository name. Within each repository directory, subdirectories are created to represent snapshots of the repository at specific points in time. Each snapshot directory is named based on the snapshot's date using the format `<YYYY-MM-DD>`.
The contents of each snapshot directory include:

- **Cloned Repository:** The actual cloned repository is stored as a subdirectory with its repository name, e.g., `CVE-2022-2588`.
- **Repository Metadata:** Metadata about the repository is retrieved using the GitHub API and stored as a JSON file named `repository_<YYYY-MM-DD>.json`.
- **User Metadata:** Information about the repository owner is retrieved using the GitHub API and stored as a JSON file named `user_<YYYY-MM-DD>.json`.

Example directory layout. An example directory layout for a repository `sniper404ghostxploit_CVE-2022-2588` with a snapshot on April 6, 2024, stored under the `CVE-2022` directory, is shown in Figure 1:
This storage design effectively organizes the collected data by grouping repositories under CVE-year directories and maintaining snapshots by date. This structure simplifies navigation and analysis, and ensures that both the repository content and associated metadata, such as user and repository information, are preserved. By uniquely identifying repositories using

```
CVE2022/
+-- sniper404ghostxploit_CVE-2022-2588/
    +-- 2024-04-06/
        +-- CVE-2022-2588/              % Cloned repository
        +-- repository_2024-04-06.json % Repository metadata
        +-- user_2024-04-06.json        % User metadata
```

Figure 1: Example of the data storage structure for GitHub repositories.

a combination of GitHub username and repository name, the design enables large-scale data collection while maintaining clarity and completeness.

5.2.5 Real-time monitoring

To gain continuous insights into the dynamics of repositories referencing CVE identifiers, a monitoring system was developed and automated using a daily cron job. The system queries the GitHub API to track changes and updates to repositories associated with CVEs from the years 2018 through 2024. The decision to start tracking from 2018 was made due to the assumption that older CVEs exhibit less activity, with fewer repositories being created or updated as time progresses. By focusing on more recent CVEs, the monitoring system captures the most relevant and evolving trends in repository engagement and exploit code development.

Daily data collection. The cron job executes a script each day, querying the GitHub API for all repositories related to CVEs in the specified range. The queries collect metadata, including the repository description, forks, stars, and other relevant details, which are saved as part of the output. This daily operation ensures that new or modified repositories are consistently monitored, maintaining an up-to-date repository record.

Tracking dynamics. Beyond collecting raw repository meta-data, the system performs analysis to monitor key statistics related to the evolution of CVE-referenced repositories. This second step involves iterating over the results from the daily queries to track:

- **Added and removed repositories:** The system logs the number of repositories added and removed for each CVE year, offering insight into repository growth and engagement dynamics.
- **CVE-ID extraction:** Basic CVE-ID extraction is performed to analyze activity and trends for specific CVEs.
- **Top statistics:** The following daily statistics are recorded to understand repository engagement patterns:
  - Added repositories per CVE year: The count of newly added repositories.
  - Removed repositories per CVE year: The count of repositories no longer available.
  - Top CVEs overall per year: The most referenced CVEs for each year.
  - Top added/removed CVEs per year: Identifies the CVEs with the highest number of repository additions or removals on a daily basis.

Insights by the monitoring system. By focusing on the dynamics of repository metadata rather than content, the monitoring system provides valuable insights into repository growth, engagement, and the evolution of exploit code over time. The data collected highlights how repository metrics such as stars, forks, and changes in the number of repositories are influenced by security vulnerabilities. The system captures trends in GitHub engagement and provides a view of how the community interacts with CVE-related content, offering an insight into the GitHub's security vulnerabilities ecosystem.

5.2.6 Data Cleaning and Preprocessing

Before conducting an analysis of the collected data, we applied a reduction procedure to streamline the analysis and eliminate redundant data for analysis. Specifically, when multiple repositories are identical, we want to retain only one unique instance for analysis. This reduction ensures our analysis remains efficient by focusing on unique repository content, while still preserving metadata such as the original user and repository name. Identical repository copies, such as those from forks, mirrors, or repeated uploads, are analyzed only once, allowing us to identify redundancy patterns and gain insight into the prevalence of duplicated content across GitHub.

To achieve this, we create a compressed archive (`.zip`) of each repository's `HEAD` (the latest commit on the default branch) and compute its `SHA-256` checksum. This hash serves as a unique identifier for the repository's content, allowing us to cross-reference and de-duplicate repositories. We note that this hashing approach has proven to be more reliable than relying on hashing the `.zip` archives that can be directly downloaded from GitHub, as these are not byte-identical for all repositories with the same contents. By generating the archive ourselves from the `HEAD` state under a consistent procedure, identical hashes reflect identical repository content.

5.2.7 CVE-ID Extraction

To support the analysis of the collected GitHub repositories, it is necessary to identify and extract Common Vulnerabilities and Exposures identifiers from the GitHub repository, since the dataset is organized around CVEs.

CVE-ID Extraction. A custom parser was developed to automatically identify and extract CVE identifiers (e.g., `CVE-YYYY-XXXX`) from repository descriptions and other metadata fields, enabling precise filtering of repositories directly associated with specific CVE entries. The following regular expression (regex) pattern was used as the basis for CVE extraction:

```
pattern = r"cve(?:-|‒|—|―|_){4}(?:-|‒|—|―|_){4,7}"
```

This pattern was applied to the repository name and description fields in an effort to extract CVE identifiers. These metadata fields were chosen initially to ensure high-confidence extraction, as the CVE references here are typically intentional and central to the purpose of the repository. After running the extraction process, it was found that some repositories did not have a CVE-ID mentioned in these metadata fields. To address this, additional fields such as the README content were parsed to uncover less prominently mentioned CVEs. These

limitations highlighted the need for manual review and further refinement, which is discussed in the following section.

Manual Review and Improvements. Following the automated extraction process, a manual investigation was conducted for the repositories where no CVE-ID was identified in the metadata fields. These repositories were examined to identify potential CVE-IDs. It was concluded that some repositories contained invalid or incorrectly formatted CVE-IDs, while others had CVE-IDs written in a format outside the scope of the regex used. Based on these findings, CVE-ID extraction was refined to improve precision in CVE extraction.
The following examples were then covered by the CVE-ID extraction:

- CVE 2016-6210
- CVE 2016 5195
- cve—2017–17562
- CVE2023-1829
- CVE20240109

Additionally, for repositories that contain more than one CVE-ID, it was observed that some repositories listed multiple CVEs because they addressed similar CVEs related to the same vendor/technology, while others contained a collection of various CVEs, such as repositories used for storing Indicators of Compromise (IOCs) or large databases of security information. This led to further adjustments in the handling of multiple repositories, ensuring that all relevant CVEs were correctly identified.

5.2.8 Repository Metadata Enrichment

The repository metadata was retrieved using the GitHub API, which provides a comprehensive set of data for each repository. Basic repository-level data including the repository name, description, creation date, last update, programming language, star and fork count, contributor information are collected and do not require further enrichment. In addition to this, user-specific information, such as creation date, follower count, is collected separately by querying the GitHub API for each repository owner. This metadata offers valuable context on the visibility, popularity status and creator context of each GitHub repository referencing a CVE.

CVE Contextualization. To further enhance the context of each CVE-repository, we focus on enriching each identified CVE with vulnerability-specific metadata as provided by the NVD[7]. For this purpose, the following fields are collected and enriched with each CVE (using NVD CVE data gathered on **July 6, 2025**):

- **CVE date and time information**
  - **published** the date and time that the CVE was published to the NVD.
  - **lastModified** the date and time when the CVE was last modified.

[7] `https://nvd.nist.gov/`

- **CVE status** the CVE's status in the NVD.
- **CVSS metrics** information on the impact of CVE, including standardized severity score and vector from CVSSv2 and/or CVSSv3[8].
- **CWE classification** the underlying weakness category (e.g., buffer overflow, cross-site scripting)[9].
- **Affected products and vendors** associated with the vulnerability according to the NVD.
- **Presence in Known Exploited Vulnerabilities (KEV) Catalog** known vulnerability that has been exploited in the wild, cataloged by CISA[10].

This CVE-centered enrichment enables a more detailed analysis of the CVE-repositories, allowing for better insight into the types of vulnerabilities being promoted in GitHub repositories, and their severity.

5.2.9 Error Handling and Rate Limiting

To ensure reliable large-scale data collection from public APIs, it is necessary to carefully handle its service constraints. During the implementation of the GitHub data retrieval pipeline, mechanisms were created to address request timeouts, incomplete responses, and unavailability. This section outlines the methods used to detect and mitigate these conditions, ensuring the consistency and completeness of the collected data set.

GitHub API Rate-limit. GitHub applies strict rate-limiting policies to regulate API access. In its default unauthenticated use, the API allows for only 60 requests per hour. Given the scale of the data, this limit was quickly exceeded during initial testing. To cope with the volume of queries necessary for extensive repository and metadata collection, authenticated access was used in combination with a personal access token. This approach increased the rate limiting to 5,000 requests per hour, enabling constant and efficient data collection without interruption.

GitHub API Pagination. In addition to rate limitations, the GitHub API also enforces a pagination constraint, returning a maximum of 100 results per page and up to 10 pages per search query, totaling to 1,000 results per query. This restriction posed a significant challenge when attempting to retrieve data exceeding this limit.
To overcome this, a query refinement strategy was developed, dynamically partitioning broader search queries into narrower, non-overlapping segments based on repository attributes. Each query was built upon on a specific CVE year and initially narrowed using the repository creation date. The first segment retrieved repositories created prior to January 1 of the CVE year, these are of particular interest and are discussed later.

Subsequently, the remaining time range was divided into monthly intervals up to the date of collection. If a monthly segment returned more than 1,000 repositories, it was further split

[8] https://www.first.org/cvss/
[9] https://cwe.mitre.org/
[10] https://www.cisa.gov/known-exploited-vulnerabilities-catalog

into two halves by date to stay within the pagination limit. All date ranges were constructed to be non-overlapping to avoid duplication. The only theoretical edge case, where more than 1,000 repositories were created on a single day, was systematically checked for across the dataset and was not encountered. Table 2 illustrates this refinement process for a partially collection repositories for CVE-Year 2021, highlighting cases where sub-querying was necessary to remain within API constraints.

This technique ensured complete data coverage while respecting the API's pagination limits and avoiding redundant results.

GitHub Data (In)completeness. While the GitHub API was used to collect repositories associated with each CVE-year, a small degree of incompleteness emerged during the subsequent data collection stage. After the initial query, which retrieved all repositories referencing a CVE-year keyword, additional steps were performed. This included fetching user metadata of the repository owner and repository metadata. As well as, cloning the repository for further analysis. Due to the small but non-negligible time interval between the initial discovery and the further data collection, a limited number of repositories (in this dataset, three known cases) became unavailable in this time.

This unavailability may be caused by repositories being deleted, made private, or restricted by the owner or by GitHub itself. Similar behavior has been observed in prior work [The22], where repositories were taken offline by GitHub due to violations of its terms of service.

The collected repositories are not guaranteed to be exclusively Proof-of-Concepts. While all repositories reference CVEs, some may not contain actual exploit code. As a result, the dataset may include non-PoC content. The implications of this are further discussed in the Limitations section.

Table 2: Refined query segments for `cve-2021` based on repository creation date

Query	DateStart	DateEnd	Total Count	Query type
cve-2021	...	...	19941	full year
cve-2021	< 2021-01-01		15	full range
cve-2021	2021-01-01	2021-01-31	239	full range
...	...	...	...	...
cve-2021	2021-06-01	2021-06-30	517	full range
cve-2021	2021-07-01	2021-07-31	1074	split (aggregate)
cve-2021	2021-07-01	2021-07-15	708	sub-query
cve-2021	2021-07-16	2021-07-31	366	sub-query
cve-2021	2021-08-01	2021-08-31	424	full range
cve-2021	2021-09-01	2021-09-30	1380	split (aggregate)
cve-2021	2021-09-01	2021-09-14	766	sub-query
cve-2021	2021-09-15	2021-09-30	614	sub-query
cve-2021	2021-10-01	2021-10-31	1138	split (aggregate)
cve-2021	2021-10-01	2021-10-15	606	sub-query
cve-2021	2021-10-16	2021-10-31	532	sub-query
cve-2021	2021-11-01	2021-11-30	713	full range
...	...	...	...	...

6 Study Results

This chapter presents a comprehensive analysis of the collected repositories. Building upon the technical methodology outlined earlier, the focus now shifts from data collection to insights derived from the collected data. Central to the analysis is the coverage of CVEs, with each repository identified and contextualized through one or more CVEs. The study examines both the structural properties and metadata of the repositories, as well as the patterns between vulnerability disclosure and repository creation dates, and shared characteristics or relationships between repositories.

Furthermore, the analysis includes quantitative summaries such as the distribution of programming languages for repositories, the number of repositories per CVE and the overall structure and activity of the repositories. These metrics provide a foundation for understanding how publicly disclosed vulnerabilities are represented and addressed within the open-source software ecosystem of GitHub. The results aim to contextualize the collected data and support broader observations regarding post-disclosure exploit development and proof-of-concept creation practices on GitHub.

6.1 Overview of Collected Repositories

Table 3 provides a summary of our data collection (snapshots) and the resulting dataset $\mathtt{D_{PoCs}}$ that we use in our study. The first snapshot ($Snapshot_1$) resulted in the collection of 88,151 repositories, which after cleaning and deduplication resulted in 18,591 unique repositories identified by their hash. As for the second snapshot ($Snapshot_2$), which resulted in collecting 5,470 repositories, of which 1,842 are unique after cleaning and deduplication.

It is important to note that the total number of 93,621 collected repositories across both snapshots does not represent the set of unique PoC repositories after cleaning and deduplication. This is due to overlapping CVE years across the snapshots, as well as the presence of repositories that reference multiple CVEs from different CVE-years. Consequently, such repositories have been collected multiple times across snapshot queries or appear in both snapshots. After merging and deduplicating the collected repositories from Snapshot_1 and Snapshot_2, the final dataset consists of 19,988 unique (87,832 non-unique) PoC repositories, highlighting the large impact of preprocessing steps for analysis. Of the 87,832 repositories collected, 13,144 are original repositories, while the remaining 74,688 repositories are forks. This distribution shows that approximately 85% of the dataset consists of forks, highlighting the large share of independent copies. This imbalance, including detailed analysis of forks types, will be examined later. As a final step in this process, a total of 301 repositories with a size of zero were excluded.

Table 3: Summary of the snapshots and the datasets

Snapshot	**Details**			
	Collected	**CVE-Years**	**# Collected repositories**	**# Repositories after cleaning and deduplication**
Initial Snapshot_1	Mar–Apr 2024	[2016–2024]	88,151	18,591
Secondary Snapshot_2	Aug 2024	2024	5,470	1,842
Dataset $\texttt{D}_{\texttt{PoCs}}$	(after merging)	[2016–2024]	93,621	19,988

6.2 Original-Fork and Unique Repository Analysis

Original repositories are typically the primary sources of proof-of-concept/exploit code and security tools, representing the initial contributions to the open-source vulnerability research ecosystem. On the other hand, forked repositories frequently arise from community engagement, either to adapt or extend existing code, or to preserve them for archival purposes.
As shown in Figure 2, the number of repositories associated with CVE years between 2016 and 2024 is largely dominated by forks. In total, approximately 85% of the repositories are forks, while only 15% represent original repositories. The distribution indicates that the majority of CVE-repositories arise from community engagement, highlighting the role of forks and collaborative contributions.

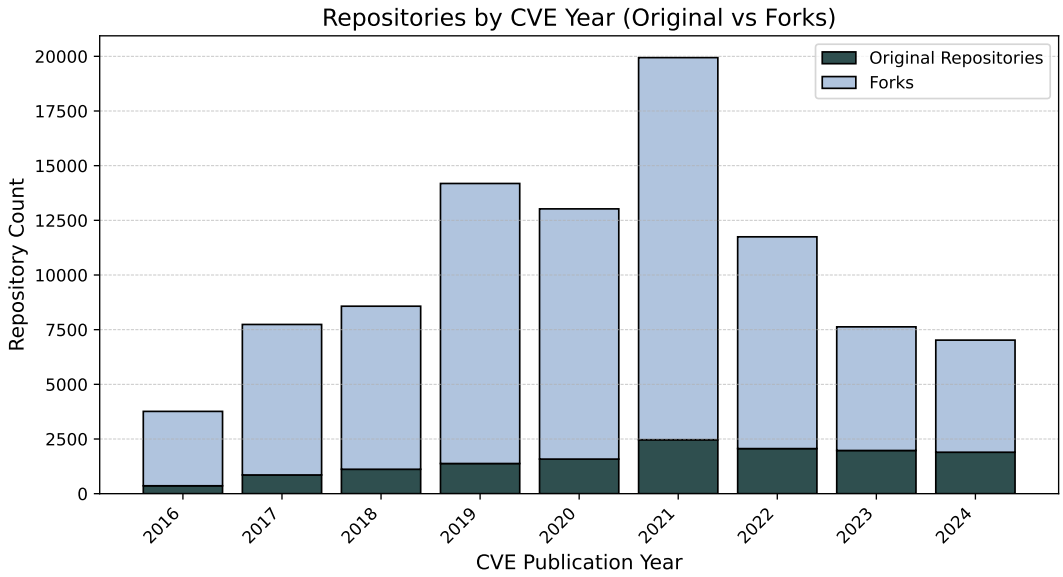

Figure 2: Distribution of collected repositories across CVE years (2016–2024), showing the proportion of original repositories versus forks.

Analysis of unique repositories, by the previously described hashing technique, reveals a clear temporal trend in repository activity across CVE publication years, as shown in Figure 3. Overall, there is a noticeable increase in unique repository/code bases over time, with distict spikes, particularly in 2021, reaching over 4,000 unique repositories. These spikes correspond to CVE-years containing high-impact vulnerabilities.

The last two CVE-year (2023 and 2024) exhibit lower counts, this is expected because the data collection occurred primarily in the first half 2024. Repositories associated with these years may still be created or updated, so the counts are likely to increase over time. In general, older CVE-years show decreasing activity after the initial surge, reflecting the natural decay of repository creation and community engagement over time.

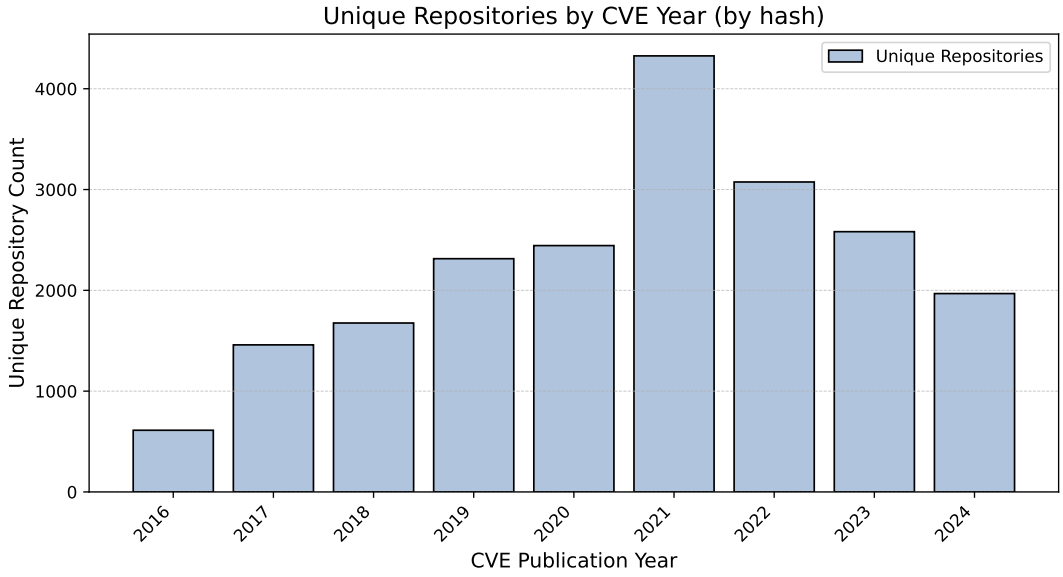

Figure 3: Distribution of unique collected repositories across CVE years (2016–2024), showing the proportion of unique repositories.

6.3 Repository Size

To characterize repository size, we use the `size` field returned by the GitHub REST API, which reports a repository's size in kilobytes.[11] Although this measure does not directly correspond to the lines of source code or executable size, it provides a consistent and comparable indicator of repository size. We use it to summarize the overall distribution of repository sizes for all collected repositories and to examine temporal variation by reporting per-year size distributions, visualized using boxplots.

Figure 4 shows a horizontal boxplot of repository sizes aggregated across all CVE years on a linear scale. The distribution is strongly right-skewed: most repositories are relatively small, while a small number of large repositories form a long upper tail. Based on manual inspection of a subset of repositories across the range of sizes, we observed that very small repositories typically consist of one or a few scripts. In contrast, substantially larger repositories frequently include non-code assets such as images, PDFs, or other auxiliary files, which contribute disproportionately to repository size. Although this pattern does not hold universally, it was a recurring characteristic among the repositories we examined.

[11] GitHub describes this field as: "The size of the repository, in kilobytes. Size is calculated hourly. When a repository is initially created, the size is 0." `https://docs.github.com/en/rest/repos/repos?apiVersion=2022-11-28#get-a-repository`

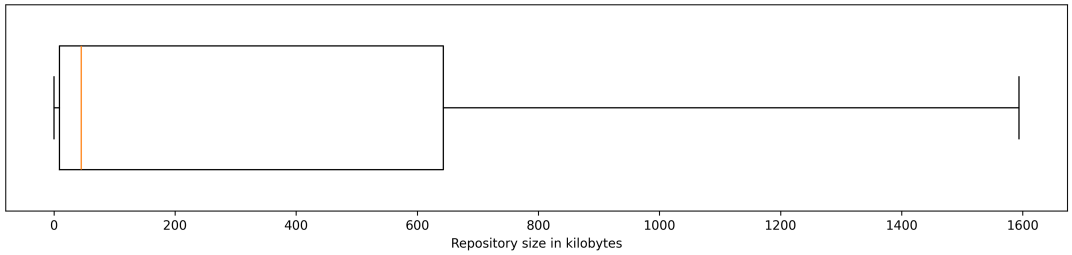

Figure 4: Distribution of repository size in kilobytes across all CVE years (2016–2024).

To assess whether this pattern is consistent over time, we further break down repository sizes by CVE year. Figure 5 presents the corresponding per-year distributions, allowing direct comparison of medians and spread across years. Across years, repository sizes remain strongly right-skewed, with broadly similar medians. However, the spread varies substantially, with marked higher variability in 2019 and 2020. The elevated variability in 2019 coincides with periods in which widely publicized vulnerabilities (e.g., BlueKeep, CVE-2019-0708) attracted extensive community attention.

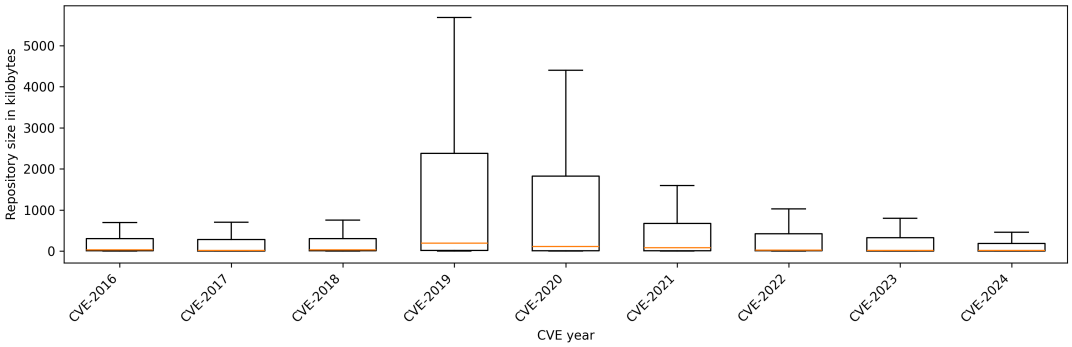

Figure 5: Distribution of repository size in kilobytes per CVE year (2016–2024).

6.4 Programming language

Each repository in our dataset is labeled with a programming language, determined as the most dominant language by GitHub's Linguist tooling[12]. This programming language is identified by GitHub's analysis of code in the repository. As such, it reflects the language that constructs the largest portion of a repository's codebase, but not necessarily the one used for exploit and proof-of-concept code. An overview of the most common programming languages (as reported by GitHub), of all collected repositories is presented in Table 4. To focus on the most prevalent languages, only those with repository counts greater than or equal to 10 are presented. A total overview of all programming languages, including those with fewer than 10 repositories, is reported in the Appendix Table 19.

Since some repositories are associated with multiple CVE years (due to references to more than one CVE-year), programming language statistics were calculated across the entire set of collected repositories, regardless of CVE-year grouping. This provides an overview of the dominant programming languages present in the dataset.

[12] `https://github.com/github-linguist/linguist/tree/main`

Table 4: Programming languages with repository counts greater than or equal to 10.

Language	Count	Language	Count	Language	Count
Unknown	62683	PHP	338	Swift	30
Python	12286	PowerShell	273	Perl	27
C	3469	Dockerfile	235	TypeScript	25
C++	1665	Objective-C	113	Makefile	21
Java	1617	Lua	89	YARA	21
Shell	1322	Batchfile	86	Rich Text Format	20
JavaScript	801	Rust	67	Jupyter Notebook	17
Go	752	ActionScript	31	Visual Basic	10
HTML	663	CSS	31	Kotlin	10
Ruby	502	Zeek	30	HCL	10
C#	485				

In our dataset, the most reported programming languages by GitHub were Python, C/C++, Java and Shell. Python's popularity results from its ease of use and extensive libraries, especially in cybersecurity and automation. C, C++ and Java remain prevalent due to their use in system-level programming. While scripting languages such as Shell and PowerShell also remain prominently, reflecting their native availability on most systems. This distribution reflects common choices for implementing proof-of-concept and exploit code in these languages, favoring rapid prototyping and broad system support [Ous98, BGL+09]. It should be noted that, for 62,683 repositories, no programming language has been labeled by GitHub's linguist tooling. This occurs because GitHub's language detection primarily reflects the unique code content of each repository. Among the 62,683 repositories, a significant proportion are unchanged forks and do not have language analysis. This was confirmed through manual inspection, which identified a large number of such unchanged forks.

Given this observation, we also report programming language statistics for original repositories. Table 5 therefore presents the language distribution of original CVE-referencing repositories. Compared to the distribution of all collected repositories, this shows slightly different results: the share of `Unknown` decreases significantly, and the relative prominence of common programming languages (e.g., Python, C/C++, Shell, and Java(Script)) becomes more apparent.

Table 5: Programming languages with repository counts greater than or equal to 10 (original repositories only).

Language	Count	Language	Count	Language	Count
Python	4262	PowerShell	195	CSS	29
Unknown	3894	Ruby	174	Zeek	25
C	981	Dockerfile	146	TypeScript	21
Java	740	PHP	145	Perl	18
Shell	675	C#	107	Makefile	15
JavaScript	389	Rust	62	YARA	14
C++	385	Lua	53	Rich Text Format	14
Go	383	Batchfile	34	Swift	13
HTML	227	Objective-C	30	Jupyter Notebook	11

To assess how representative GitHub's reported primary programming language is for the actual implementation language of PoC code, we conducted a small-scale validation through random sampling. Specifically, we selected the ten most frequent GitHub-reported programming languages in our dataset (Python, C, C++, Java, Shell, JavaScript, Go, HTML, Ruby, and C#) and randomly sampled fifteen repositories for each language, resulting in a total of 150 repositories. For each sampled repository, we manually inspected its contents to identify files associated with PoC behavior, such as exploit scripts or proof-of-concept code, rather than auxiliary files or documentation.

We note that while all sampled repositories reference one or more CVEs, not all repositories contain executable exploit or proof-of-concept code; some repositories provide analysis, documentation, or auxiliary tooling. In such cases, the repository was labeled as having no identifiable PoC implementation language. Across the 150 manually inspected repositories, GitHub's reported primary language matched the observed PoC implementation language in 124 cases (82.7%), whereas in the remaining 26 (17.3%) cases the reported primary language did not align with the observed PoC implementation language, as summarized in Table 6. Among the 26 cases, most mismatches (20 out of 26) were primarily the case due to CVE-referencing repositories that did not contain PoC or exploit code (e.g., fixes, (vulnerable) test applications, or honeypots), while a smaller subset (6 out of 26) contained PoC/exploit code implemented in a different language than the repository's dominant codebase.

Table 6: Validation of GitHub-reported primary language.

GitHub Language	**Samples**	**Match**	**No / Other PoC**
Python	15	15	0
C	15	12	3
C++	15	14	1
Java	15	10	5
Shell	15	14	1
JavaScript	15	9	6
Go	15	14	1
HTML	15	11	4
Ruby	15	15	0
C#	15	10	5
Total	150	124	26

Overall, this validation supports the use of GitHub's reported primary language as a reasonable basis for interpreting the repository-level language statistics.

6.5 CVE-Identifier

The repository collection was initiated by CVE-year, leveraging search terms based on CVE identifiers corresponding to these CVE-years. To systematically analyze the CVE-identifiers mentioned in our dataset, the custom parser, as described in Section 5.2.7, was used to automatically identify and extract CVE identifiers. This extraction enabled detailed examination of the CVEs linked to each repository, providing insights beyond the initial CVE-year grouping. This analysis focuses on the most prominent CVEs found across the collected repositories, the distribution of repositories per CVE, and the nature of the targeted vulnerabilities.

A total of 6,531 unique CVEs were extracted from all collected repositories. Figure 7 illustrates the distribution of CVEs by number of associated repositories. For readability, only the 20 CVEs with the highest repository counts are displayed. This reflects both the historical impact and the ongoing relevance of certain vulnerabilities. CVEs, such as CVE-2021-44228 (Log4Shell)[13] and CVE-2019-0708 (BlueKeep)[14], have received sustained attention due to their high severity and the urgency reflected in prominent advisories [Cyb21, Cyb19].

Many of the prominent CVEs correspond to remote code execution (RCE) vulnerabilities affecting widely used software, such as Java applications, Microsoft Windows, and Apache services. This aligns with the observed trends of the information security community to focus on vulnerabilities that are both easily exploitable, highly impactful, and common in real-world systems are more frequently discussed in the context of active attacks, security advisories and reports [ENI25]. Vulnerabilities such as CVE-2020-0796 (SMBGhost)[15] and CVE-2020-1472 (Zerologon)[16] illustrate the concentration around vulnerabilities in critical network protocols and authentication mechanisms. The high prevalence of repositories related to older CVEs, such as CVE-2016-5195 (Dirty COW)[17], highlights the interest in vulnerabilities that remain relevant for legacy systems. This suggests that repository counts may reflect not only immediate post-disclosure activity but also sustained, long-term engagement with CVEs on GitHub.

It is important to note that the repository counts may be influenced by unrelated factors other than the technical characteristics of a CVE. In particular, vulnerabilities that receive substantial public or media attention are often associated with increased community activity, including a larger number of GitHub repositories referencing these vulnerabilities.

Table 7: Top 20 CVEs by associated repository count.

CVE Identifier	**Count**	**CVE Identifier**	**Count**
CVE-2021-44228	4669	CVE-2019-6340	1148
CVE-2019-0708	2597	CVE-2016-5195	1076
CVE-2020-1938	1605	CVE-2020-10199	1037
CVE-2020-2551	1548	CVE-2020-1472	1032
CVE-2021-4034	1546	CVE-2019-17558	957
CVE-2020-0796	1463	CVE-2020-10204	952
CVE-2020-14882	1421	CVE-2020-11444	936
CVE-2020-2883	1187	CVE-2017-11882	927
CVE-2021-1675	1182	CVE-2021-40444	895
CVE-2020-2555	1166	CVE-2021-3156	831

For a better understanding of our PoC dataset and the targeted vulnerabilities, we cross-checked the extracted CVE identifiers against the NIST National Vulnerability Database (NVD) to determine how many CVEs have at least one corresponding GitHub repository in our dataset. The results are summarized in Table 8. We report both the initially extracted CVEs and the *validated* CVEs after NVD verification; this step filters out malformed or non-existent identifiers

[13] `https://nvd.nist.gov/vuln/detail/cve-2021-44228`
[14] `https://nvd.nist.gov/vuln/detail/cve-2019-0708`
[15] `https://nvd.nist.gov/vuln/detail/cve-2020-0796`
[16] `https://nvd.nist.gov/vuln/detail/cve-2020-1472`
[17] `https://nvd.nist.gov/vuln/detail/cve-2016-5195`

(e.g., misspellings or incorrect CVE strings) that appear in repository metadata. Overall, only a small fraction of NVD-listed CVEs are referenced by GitHub repositories, with yearly coverage typically on the order of a few percent. This suggests that publicly available proof-of-concept and exploit code on GitHub concentrates on a limited subset of disclosed vulnerabilities rather than covering the full set of reported CVEs.

While the absolute number of referenced CVEs generally increases over time, coverage relative to the total number of NVD entries remains broadly stable across most years. An exception is 2024, where the coverage percentage is lower. This is explained by data collection timing: our GitHub repository snapshot for 2024 was collected earlier, whereas the NVD data was retrieved in 2025, after additional CVE entries for 2024 had become available. Consequently, the denominator for 2024 is larger than at the time of repository collection, which reduced the observed coverage.

Table 8: CVEs referenced by GitHub repositories before and after NVD validation, relative to total NVD entries per year.

Year	Extracted CVEs	NVD Entries	Validated CVEs (NVD %)
2016	242	10,560	201 (1.90)
2017	408	17,024	378 (2.22)
2018	552	17,495	524 (3.00)
2019	748	17,082	681 (3.99)
2020	888	20,641	839 (4.07)
2021	967	23,095	902 (3.91)
2022	976	26,909	913 (3.39)
2023	1,122	30,158	1,061 (3.52)
2024	763	38,675	711 (1.84)

Table 9 presents the distribution of vulnerability types in our GitHub PoC dataset. We categorize CVEs by their associated weakness information (CWE), as provided in the NIST National Vulnerability Database (NVD). Each row corresponds to a weakness category and reports the number of CVEs in our dataset mapped to that category, providing an overview of the security issues most frequently targeted by public PoCs. Overall, the collected PoCs cover a broad range of weakness types, with code-execution-related weaknesses appearing most frequently. Followed by injection and memory-safety issues, while more than half of the CVEs fall into a long tail of less frequent CWE categories.

Table 9: Top CWE categories associated with CVEs referenced by GitHub PoCs. "Other CWEs" aggregates all remaining CWE categories outside the top list.

CWE Category	**Count**
CWE-79 (Cross-Site Scripting)	834
CWE-89 (SQL Injection)	400
CWE-22 (Path Traversal)	391
CWE-787 (Out-of-Bounds Write)	388
CWE-78 (OS Command Injection)	310
CWE-20 (Improper Input Validation)	276
CWE-94 (Code Injection)	241
CWE-502 (Unsafe Deserialization)	229
CWE-434 (Unrestricted File Upload)	222
CWE-416 (Use-After-Free)	209
Other CWEs	3,653

When examining original and forked repositories separately, clear differences emerge in the distribution of CVEs. Table 10 compares the CVEs referenced in original repositories to those referenced in forked repositories, displaying the 20 CVEs with the highest repository counts for readability. The distribution of CVEs mirrors the earlier repository count results in Figure 2, with forked repositories contributing to the majority of repositories per CVE year. This again highlights the higher level of community engagement in forked repositories relative to original repositories.

Table 10: Top 20 CVEs by associated repository count, separated by original and forked repositories.

Original repositories		Fork repositories		
CVE Identifier	**Count**	**Mut.**	**CVE Identifier**	**Count**
CVE-2021-44228	408	→	CVE-2021-44228	4261
CVE-2021-4034	159	↑	CVE-2019-0708	2468
CVE-2019-0708	129	↑	CVE-2020-1938	1572
CVE-2021-41773	117	↑	CVE-2020-2551	1537
CVE-2022-0847	98	↓	CVE-2021-4034	1387
CVE-2020-0796	92	↑	CVE-2020-14882	1385
CVE-2024-6387	92	↓	CVE-2020-0796	1371
CVE-2018-6574	90	↑	CVE-2020-2883	1177
CVE-2022-30190	83	↑	CVE-2020-2555	1155
CVE-2022-22965	78	↑	CVE-2019-6340	1137
CVE-2021-3156	74	↑	CVE-2021-1675	1134
CVE-2017-5638	72	↑	CVE-2020-10199	1030
CVE-2022-26134	71	↑	CVE-2016-5195	1023
CVE-2020-1472	69	→	CVE-2020-1472	963
CVE-2022-1388	64	↑	CVE-2019-17558	953
CVE-2024-3094	63	↑	CVE-2020-10204	947
CVE-2024-32002	63	↑	CVE-2020-11444	932
CVE-2020-5902	58	↑	CVE-2017-11882	893
CVE-2022-22947	56	↑	CVE-2021-40444	858
CVE-2016-5195	53	↑	CVE-2017-10271	798

Mut. indicates rank mutation when comparing the top-20 lists: ↑ higher rank in forks or newly enters the fork top-k, ↓ lower rank in forks, → unchanged.

To allow for a meaningful comparison, the counts for both groups were normalized to relative frequencies. For each CVE i, the normalized values were calculated as follows:

$$f_i^{(o)} = \frac{c_i^{(o)}}{\sum_j c_j^{(o)}}, \qquad f_i^{(f)} = \frac{c_i^{(f)}}{\sum_j c_j^{(f)}} \tag{1}$$

where $c_i^{(o)}$ and $c_i^{(f)}$ denote the counts of each CVE i in the original and forked repositories, and $f_i^{(o)}$ and $f_i^{(f)}$ represent their corresponding normalized frequencies.

Figure 6 shows the normalized frequency of CVE mentions in original repositories (x-axis) against those in forks (y-axis). Points close to the diagonal indicate CVEs for which forks follow originals proportionally, whereas points above the diagonal reveal CVEs that receive disproportionately more attention from the community through forking. Vice versa, points under the diagonal reveal CVEs that receive more attention in original repositories. Notably, CVE-2021-44228 (Log4Shell) and CVE-2019-0708 (BlueKeep) appear prominently above the diagonal, reflecting their a high number of references across forked repositories.

To evaluate whether the distributions of CVE mentions differ between original and forked repositories, a paired non-parametric Wilcoxon signed-rank test was applied. This test is appropriate for the paired numerical observations of CVEs that do not necessarily follow a normal distribution. For each CVE there is a pair of data, with corresponding counts for both original repositories as well as forked repositories. All assumptions for the test are satisfied, as the data consist of paired numerical counts for each CVE, with meaningful differences that can be ranked.

The Wilcoxon signed-rank test revealed a highly significant difference between the normalized CVE counts in original and forked repositories ($W = 3{,}777{,}092$, $p < 0.001$),
These findings indicate that the distribution of CVE mentions in forked repositories differs from the original repositories.

Although the original repositories reflect the initial contributions of the original authors, forks capture broader community engagement, highlighting what CVEs attract attention. Together, this distinction potentially indicates that original repositories and their forks may serve different roles in the dissemination of vulnerability information.

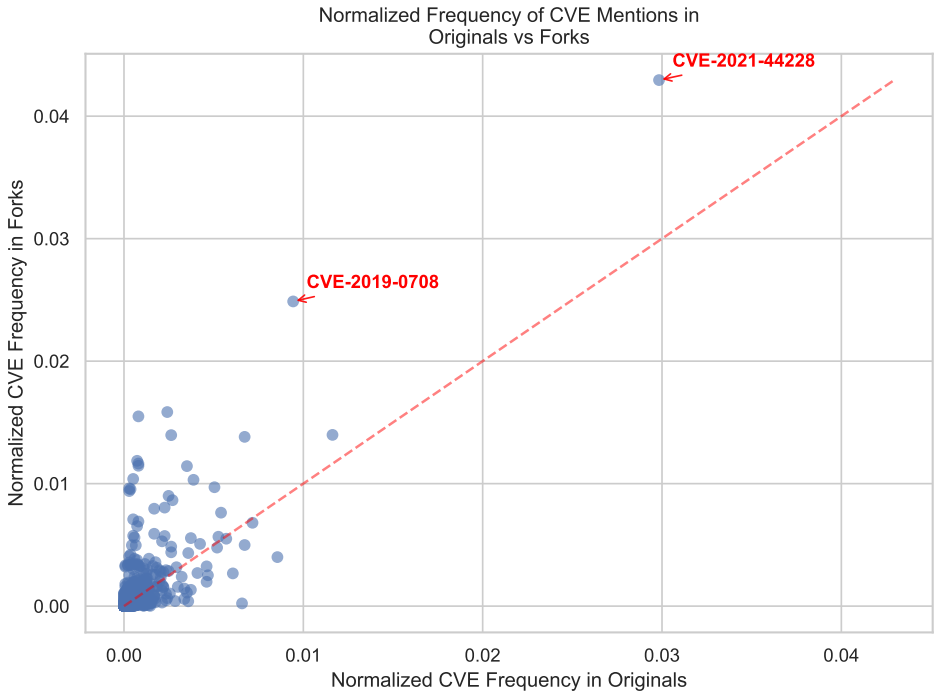

Figure 6: Comparison of normalized CVE mentions in original repositories versus forks.

From the 6,531 unique CVEs extracted from all collected repositories, 5,775 ($\approx 88.4\%$) CVEs were referenced in original repositories, indicating that original repositories account for the majority of tracked CVEs. Forked repositories reference 3,540 ($\approx 54.2\%$) CVEs, reflecting subsequent community engagement. This proportion supports the view that original repositories play a foundational role, with forks amplifying community engagement of selected CVEs.

Figure 7 illustrates the overlap between the two sets. The intersection contains 2,784 ($\approx 42.6\%$ from all identified CVEs) CVEs, indicating vulnerabilities that are represented in both original and forked repositories. Out of these 2,784 CVEs, NVD metadata was available for 2,715 entries. For the 5,775 CVEs referenced in the original repositories, NVD metadata was available for 5,412 CVEs.

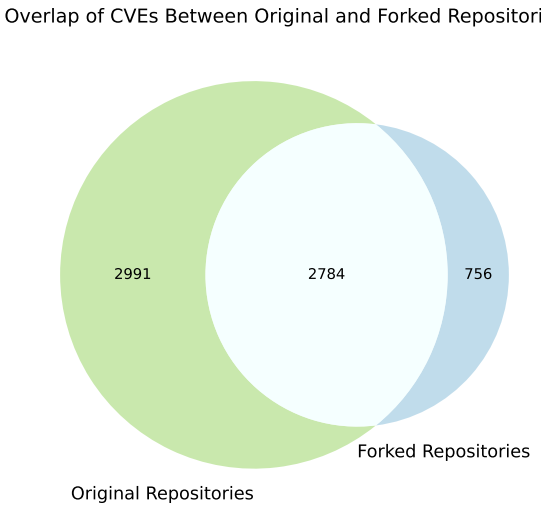

Figure 7: Venn diagram illustrating the overlap of unique CVEs between original and forked repositories. Circle areas are proportional to set sizes.

The analysis of the metadata of these CVEs is shown in Table 11, split between original and forked repositories. The results show both the raw statistics, and the weighted statistics, adjusted by repository counts to account for differences in CVE-repository volumes. The comparison shows that the CVEs referenced in forked repositories have a slightly higher mean CVSS score (7.96 vs. 7.72 and 8.55 vs. 8.26 weighted), and a larger share of High and Critical vulnerabilities, suggesting selective propagation of more impactful vulnerabilities through forked repositories. In both groups, nearly all vulnerabilities are exploitable through network or local vectors ($\approx$ 97/98%), emphasizing their relevance to remotely accessible systems. The high prevalence of CVEs with low attack complexity further suggests limited complexity to exploit vulnerabilities in practice.

Overall, both groups predominantly reference severe, accessible, and easily exploitable CVEs, indicating that forks reinforce the visibility and reuse of impactful vulnerabilities within the community.

Table 11: Comparison of CVEs referenced in original and forked repositories. Metrics are shown both unweighted and weighted by repository count.

Metric	Unweighted	Weighted
Original Repositories		
Mean CVSS Score	7.72 (σ=1.65)	8.26 (σ=1.55)
High / Critical Severity	71.3% (45.9 / 25.4)	82.6% (43.4 / 39.2)
Network / Local Exploitable	97.3% (75.9 / 21.4)	97.8% (75.9 / 21.9)
Low Attack Complexity	91.2%	90.8
Forked Repositories		
Mean CVSS Score	7.96 (σ=1.57)	8.55 (σ=1.37)
High / Critical Severity	77.8% (48.7 / 29.1)	90.8% (46.0 / 44.8)
Network / Local Exploitable	97.3% (73.4 / 23.9)	98% (73.7 / 24.3)
Low Attack Complexity	88.9%	87.7%

Among the 756 CVEs, appearing in a total of 2400 repositories, that are exclusively referenced in forked repositories and not in original repositories, we conducted a manual review to better understand their presence. From the 2400 repositories, we identified 1068 repository owners. Indicating that the distribution of repositories per owner is skewed, with a single repository owner having 517 repositories. The median number of repositories per owner is only 1, indicating that the behavior is driven by a small set of users.

An example is shown in Figure 8, which illustrates the fork tree of a repository originally named CVE-2017-10271. Several forks of this repository were found to have been renamed to CVE-2017-10272, while the contents of the forks are identical.

Figure 8: Fork Tree showing forked repositories with slightly changed CVEs.

To better understand this pattern, we manually examined a representative subset of 50 randomly selected owners of these forked repositories and categorized these cases of modified CVE identifiers as follows.

- **Related CVE identifiers** The modified CVE corresponds to a vulnerability affecting the same product or technology as the original (e.g., consecutive CVE identifiers assigned by a CNA). These cases remain relevant to the repository contents.
- **Unrelated or invalid identifiers:** The modified CVE differs by year, refers to an unrelated vulnerability, or is invalid, and therefore are not relevant to the repository contents.

Table 12 summarizes the distribution of these categories. From the 50 repository owners, spanning across 70 repositories, most of the modified CVE identifiers correspond to related CVE identifiers, often consecutive, suggesting that repository owners intentionally rename repositories to capture exploits for similar technologies. While a bit less prominent, repositories with unrelated or non-existing CVEs reflect misuse or use as a placeholder.

Table 12: Categorization of modified CVE Identifier in Forked Repositories

Category	**Count**
Related CVE identifiers	46
Unrelated or invalid identifiers	24
Sample Size	70

Manual inspection of the repositories contents further supports this interpretation. In many of these cases, forked repositories contain identical code to the original repository, with only small modifications to the referenced CVE in the repository name or description. This pattern indicates that forked repository owners aim to differentiate the repository name of the fork. However, for a smaller subset of repositories, we observed legitimate additions to the code, where owners of the repository are trying to capture similar exploits.

6.5.1 CVE-Release-Repo-Creation-Date

To better understand the dynamics of CVE publishing and exploit development, we analyze the relationship between repository creation dates and the disclosure dates of its referenced CVEs. This comparison provides insight into how quickly proof-of-concept repositories are created relative to the official vulnerability publication. To ensure a clear mapping between repository and vulnerability, we restrict our analysis to repositories that reference a single CVE identifier. This eliminates ambiguity from repositories referencing multiple CVEs, where the creation date may not consistently correspond to a specific CVE. For the disclosure dates of CVEs, we made use of the National Vulnerability Database (NVD) metadata. Specifically, we used the NVD `published` timestamp as the reference date for when a CVE is publicly disclosed in a centralized, widely used catalog. We use this field because it is consistently available across CVEs, and commonly used as a standardized reference point.

From the total of 87,832 repositories from distinct owners, 76,741 repositories reference a single CVE identifier, where 11,091 repositories reference multiple CVEs. This subset of repositories referencing a single CVE represents the majority of repositories, and is used for this analysis. To improve interpretability, repositories created more than one year before or three years after the corresponding CVE publication date were excluded from this analysis bringing the total to 71,457 repositories for this analysis. The cutoff makes sure, focus is on the period

where CVE-related repository creation is most likely to occur, reducing the influence of outliers.

The plot in Figure 9 visualizes the distribution of the time difference between CVE release dates and repository creation dates. The x-axis represents the number of days between the release of a CVE and the creation of a repository associated with it. The y-axis shows the number of repositories created during each specific time frame. The figure shows this distribution, where repository creation drops off as more time passes after the publication date of a CVE. This decreasing frequency over time suggests natural decay, as after disclosure, patches are released or newer vulnerabilities emerge.

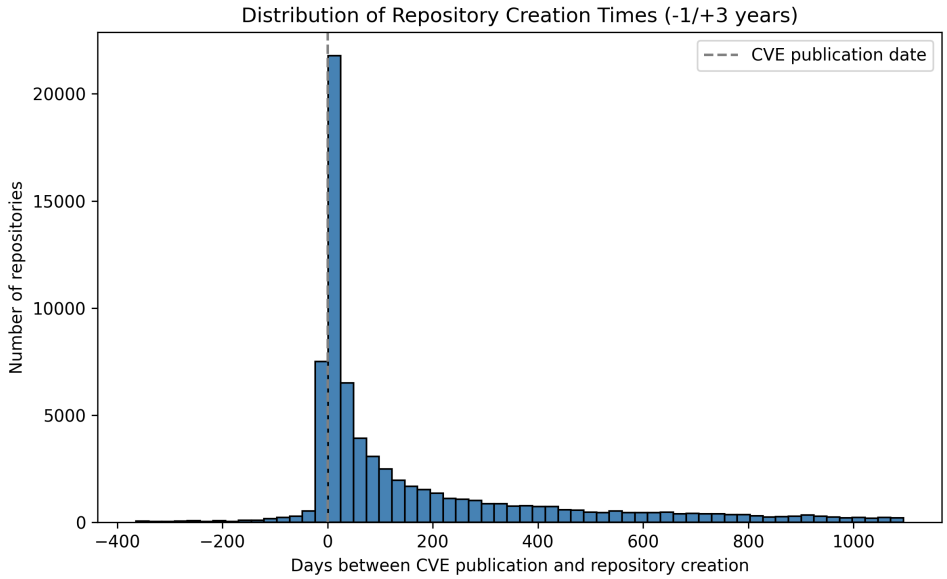

Figure 9: Distribution of repository creation times relative to CVE release dates, showing the number of repositories created at different time intervals between CVE release and repository creation.

Table 13 shows repository creation over time following CVE publication. Within the first day, roughly 8% of repositories are created, reflecting immediate attention. Within five days, this grows to 18%, and by 30 days, nearly 40% of repositories have appeared. Notably, more than half of all repositories (55.7%) are created within the first 100 days, indicating that attention is concentrated early but continues to accumulate over several months. After one year, 76.4% of repositories have been created, suggesting a long tail of continued activity.

Table 13: Cumulative percentage of repositories created after CVE publication over time.

Days after CVE	Cumulative % of Repositories
1	7.96%
5	18.03%
10	25.30%
30	39.13%
50	45.70%
100	55.72%
180	64.94%
365	76.37%

Due to the long-term decline in activity following a CVE publication, we narrowed our analysis to a more narrow window surrounding the release date. This closer view highlights the short-term dynamics after a vulnerability publication. Based on the distribution of repository creation and activity in our dataset, we focus on the period from 10 days before to 30 days after publication, as activity beyond this window stabilizes. Figure 10 illustrates this distribution, which likely reflects the attention of developers and researchers. As observed, repository creation peaks shortly after publication and gradually decreases over time, showing a clear decay pattern in activity within this critical window.

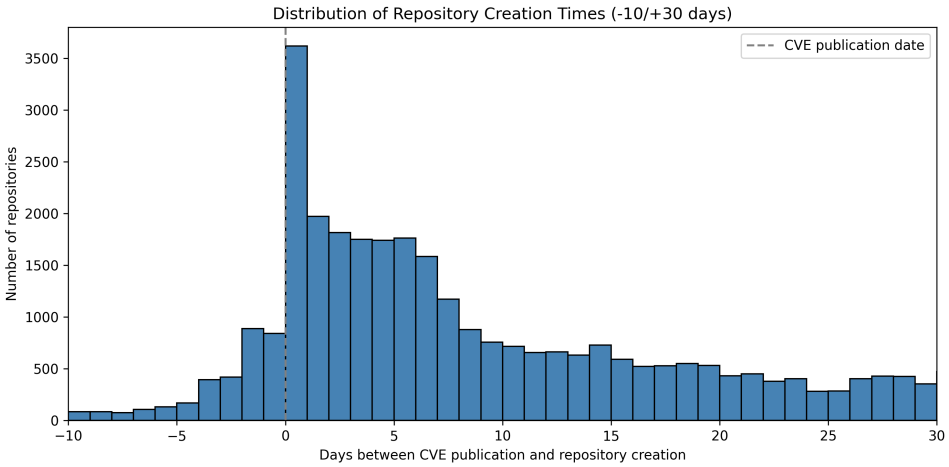

Figure 10: Distribution of repository creation times relative to CVE release dates, showing the number of repositories created at different time intervals between CVE release and repository creation.

Interestingly, a portion of the repositories are created before the official CVE publication date. A total of 6,045 repositories accounting for 7.93% of the total collected repositories are created before the publication date of the referenced CVE. This suggests that some vulnerabilities attract attention before the NVD's publication of the CVE. Since we use the NVD as the single source of truth, because it captures all CVEs, it is possible for CNAs to publish CVE entries earlier. As a result, a CVE may already have been made public before it is published by the NVD. Additionally, it could be the case due to early information sharing or existing GitHub repositories later referencing these CVE identifiers. Together, this explains why some GitHub repositories appeared before the NVD published the CVE.

As shown in Figure 11, when examining activity up to 100 days before publication, overall engagement remains consistently low. However, approximately 20 days before CVE publication, we observe a slight increase. Notably, on the days preceding CVE publication, the number of created repositories rises to around 6,000, suggesting that activity around these vulnerabilities often emerges shortly before its official disclosure by the NVD.

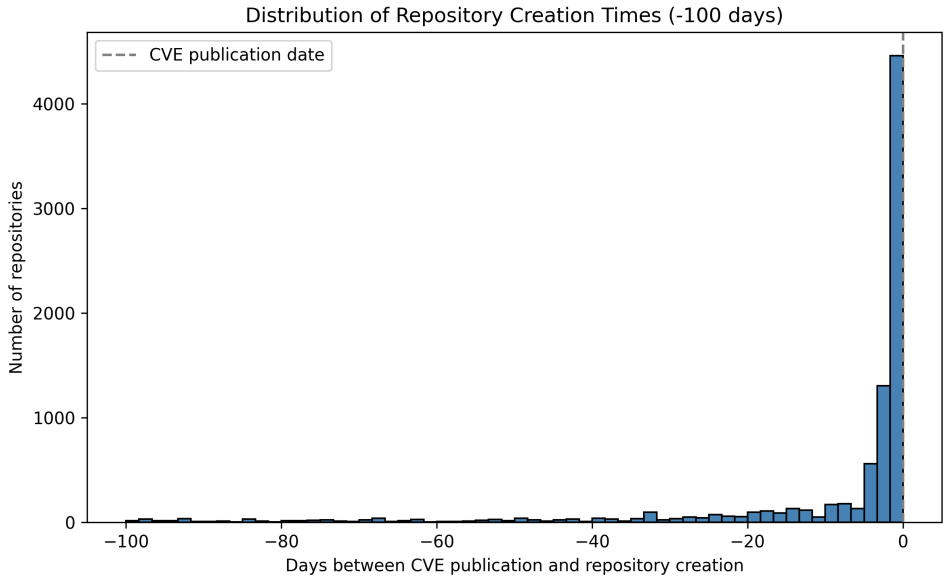

Figure 11: Distribution of repository creation times relative to CVE release dates, showing the number of repositories created at different time intervals between CVE release and repository creation.

Because the CVE publication dates are retrieved from the NVD database, it is possible that certain CVEs are referenced by their assigning CNAs before the official NVD release date. By examining the associated CNAs (Table 14), we observe that only 27,4% (1656 repositories) of the CVEs were assigned by the NVD (via cve@mitre.org). The remaining 78.2% of repositories correspond to other CNAs, potentially explaining the activity observed prior to the official publication date by the NVD.

Table 14: Number of repositories per CVE Numbering Authority (CNA).

CNA	Count	CNA	Count
cve@mitre.org	1656	vultures@jpcert.or.jp	5
secalert@redhat.com	1016	security-alert@hpe.com	5
secure@microsoft.com	670	cret@cert.org	5
chrome-cve-admin@google.com	427	security@acronis.com	5
product-security@apple.com	383	bressers@elastic.co	5
security@apache.org	287	psirt@amd.com	5
security@ubuntu.com	227	openssl-security@openssl.org	4
psirt@fortinet.com	110	psirt@nvidia.com	4
security@php.net	108	report@snyk.io	4
security@vmware.com	107	twcert@cert.org.tw	4
audit@patchstack.com	94	security@opentext.com	3
security@android.com	87	security@huntr.dev	3
psirt@lenovo.com	69	hp-security-alert@hp.com	3
security@mozilla.org	65	cna@sap.com	3
support@hackerone.com	65	secteam@freebsd.org	3
psirt@adobe.com	63	psirt@honeywell.com	3
zdi-disclosures@trendmicro.com	60	security@duo.com	2
contact@wpscan.com	50	mlhess@drupal.org	2
secalert_us@oracle.com	42	secure@symantec.com	2
f5sirt@f5.com	36	security@trendmicro.com	2
psirt@cisco.com	35	product-security@qualcomm.com	2
cve@gitlab.com	32	psirt@purestorage.com	2
security-advisories@github.com	30	0fc0942c-577d-436f-ae8e-945763c79b02	2
security@wordfence.com	29	productcert@siemens.com	1
ics-cert@hq.dhs.gov	23	emo@eclipse.org	1
secure@intel.com	19	cve-request@iojs.org	1
security@unisoc.com	19	CybersecurityCOE@eaton.com	1
security@atlassian.com	18	vuln@vdoo.com	1
psirt@huawei.com	15	cve-requests@bitdefender.com	1
cve-coordination@incibe.es	14	trellixpsirt@trellix.com	1
cna@vuldb.com	12	psirt@zte.com.cn	1
patrick@puiterwijk.org	11	security@joomla.org	1
security@debian.org	10	info@cert.vde.com	1
jordan@liggitt.net	10	security@mediatek.com	1
security@qnapsecurity.com.tw	10	security@zyxel.com.tw	1
cve-coordination@google.com	8	product.security@lge.com	1
secure@citrix.com	8	biossecurity@ami.com	1
security_alert@emc.com	6	416baaa9-dc9f-4396-8d5f-8c081fb06d67	1
security-officer@isc.org	6	psirt@esri.com	1
psirt@us.ibm.com	6	security-report@netflix.com	1
cve@rapid7.com	6	df4dee71-de3a-4139-9588-11b62fe6c0ff	1

6.6 Dynamics via Monitoring system

While the static snapshot provides a comparison between repository creation times and CVE publication dates, our monitoring system extends this analysis. The system tracks repositories referencing CVE identifiers as they are created and disappear over time, enabling real-time observation of vulnerability-related activity across GitHub. This allows us to capture emerging CVE mentions before, during and after disclosure, and to observe how engagement evolves. In contrast to the static dataset, the monitoring view highlights the dynamic nature of CVE-related development activity in the wild.

We evaluate the completeness of the daily monitoring system (Table 15) by checking whether all expected year buckets (`CVE-2018` till `CVE-2024`) are present for each monitoring date. In total, 477 dates satisfy this completeness, spanning from 2024-04-30 to 2025-11-30. Over this interval, daily monitoring would have expected 580 days to be monitored. However, 103 days (17.76%) did not produce a complete daily snapshot, as not all data for each CVE-year was gathered. For this reason, we excluded these from subsequent analysis. These gaps are primarily attributable to constraints of the GitHub API, including rate limiting and availability issues, which prevented completing all queries within a given day. To mitigate this, the monitoring system was iteratively improved during the study period, by adjusting the scheduling and speed of the monitoring. The largest gap is 22 days, occurring between 2024-08-01 and 2024-08-23. Figure 12 visualizes daily coverage and highlights periods without collected data.

Table 15: Coverage statistics for the daily monitoring system.

Metric	Value
Complete monitoring dates (`CVE-2018`–`CVE-2024`)	477
Oldest complete date	2024-04-30
Newest complete date	2025-11-30
Range span (days)	580
Missing days	103 (17.76%)
Largest gap (days)	22
Largest gap interval	2024-08-01 → 2024-08-23

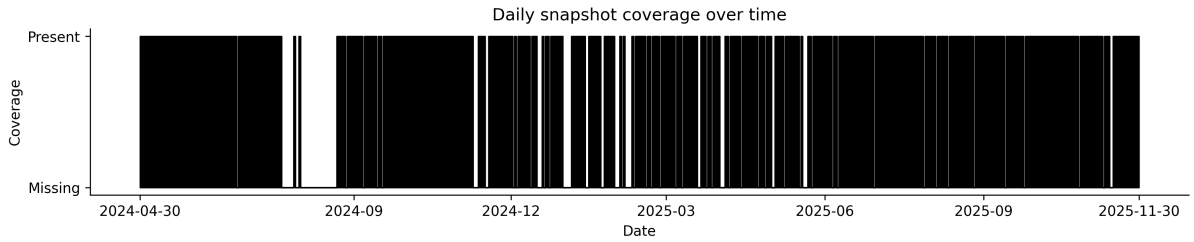

Figure 12: Daily monitoring coverage visualization over time.

Beyond coverage, the daily monitoring system enables a dynamic view of repository availability across the CVE-years. For each complete day, we aggregate per-year search results and analyze:

(i) year-level composition (shares per CVE-year),

(ii) overall volume over time,

(iii) day-to-day creation and deletion of repositories.

This provides a coarse-grained view of how exploit-related repository activity evolves and how attention shifts between older and newer CVE-years.

In an initial visualization of the monitoring time series, we observe occasional drops in the reported repository counts in one or more CVE-year buckets (Figure 13). Since incomplete days with zero results were filtered out beforehand, these drops reflect incomplete snapshots where only a subset of repositories were retrieved due to encountered API limitations. Given that the observed decreases are followed by an immediate recovery in the next snapshot, they are more likely to be caused by partial collection than with actual changes in repository counts. As the monitoring system is intentionally paced and executed over an extended time window (often exceeding 12 hours) to remain within GitHub API limits, we do not perform automatic retries for partial results on the same-day. Therefore, under-collection on a single given day is possible in the monitored window.

The figure highlights two types of drops. First, the red markers indicate isolated drops in a single CVE-year bucket, suggesting partial failure of a specific year query. Second, the blue marker corresponds to a simultaneous decrease across all CVE-year buckets, indicating a broader collection failure impacting the full monitoring.

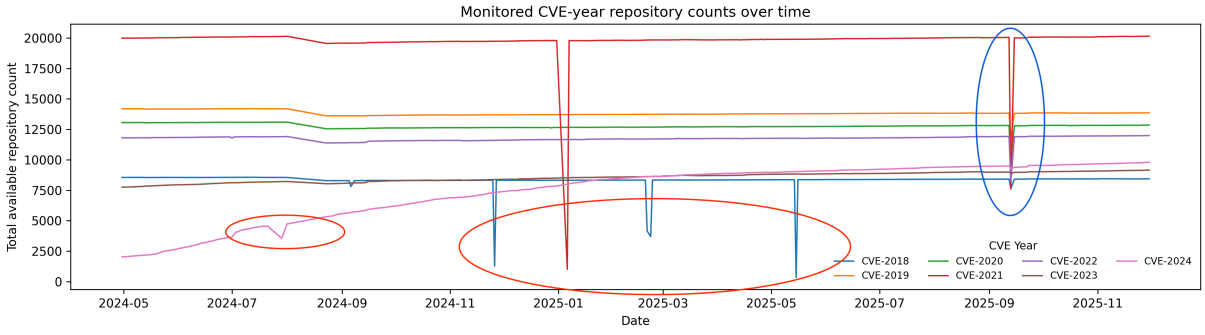

Figure 13: Daily monitored repository counts over time.

To reduce the influence of such artifacts while preserving meaningful longer-term insights (including real deletions), we apply a conservative cleaning step to the time series data. We flag days that show an abrupt drop followed by a rebound within the next few days and adjust these points via interpolation between neighboring repository counts.

Figure 14 shows the cleaned series used in the remainder of this section. Older CVE-years, such as `CVE-2019` and `CVE-2021`, consistently account for the largest share of repositories throughout the monitoring window, whereas `CVE-2024` starts from a much smaller baseline and shows the clearest relative growth over time. This reflects that newly disclosed CVEs accumulate PoC repositories progressively, while older CVE-year collections are stable and change more gradually.

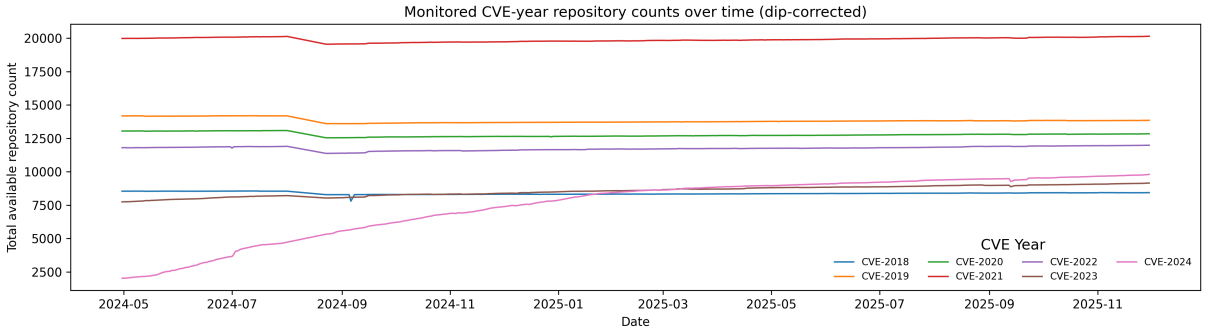

Figure 14: Daily monitored repository counts over time, with dips corrected.

Finally, to summarize overall monitoring volume and changes in year-level composition, Figure 15 reports a total view of repository availability over time (stacked by CVE-year), enabling comparison of both total volume and the relative share of each CVE-year. The total monitored repository volume remains relatively constant over the monitored period, while the composition shifts as newer CVE-year buckets increase their volume. In particular, the expanding volume corresponding to `CVE-2024` indicates growing activity in the most recent year, whereas earlier CVE-years keep their dominant share to the overall volume.

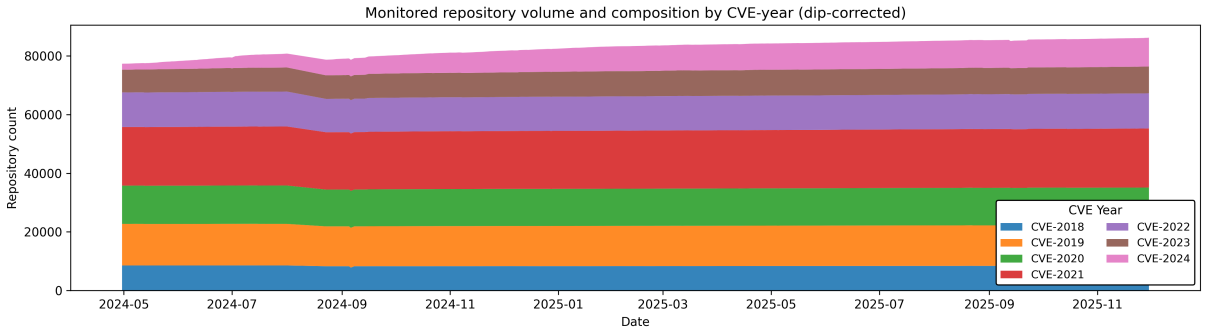

Figure 15: Daily monitored repository counts stacked over time, with dips corrected.

While the previous analysis aggregates repository availability by CVE-year, the monitoring system also enables tracking repository events, such as creations and deletions of repositories. Each monitored repository is identified by its GitHub `repo_id` and associated metadata, including the repository creation timestamp (`created_at`). We use this field to identify repositories that were newly created during the monitoring window. By comparing `created_at` to the monitoring date, we can quantify the arrival of newly created CVE-referenced repositories over time and study the variation in creation activity across different CVE-years.

Figure 16 shows the cumulative number of repositories created per CVE-year during the monitoring window. As expected, `CVE-2024` shows the largest growth, showing substantially more newly created repositories than earlier years. The `CVE-2023` bucket shows the second-highest growth, followed by earlier years with progressively smaller increases. Overall, the figure indicates that PoC publication and repository creation activity is concentrated on more recent CVE years, while older years show limited new repositories during the monitored interval.

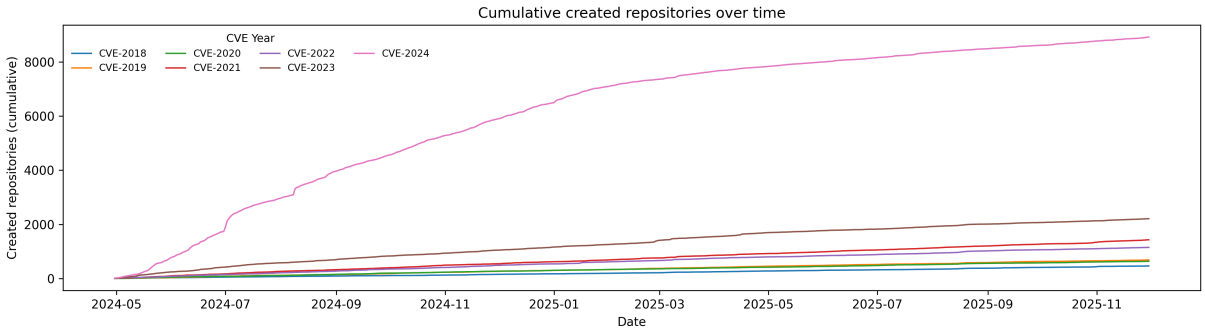

Figure 16: Daily monitored repository creations over time.

In addition, the longitudinal nature of the monitoring data allows us to approximate repository deletions or disappearances. If a previously observed `repo_id` is no longer present in subsequent daily snapshots, we interpret this as a removal event, which may correspond to repository deletion, renaming, privatization, or changes in search visibility. Although this does not identify the cause of disappearances from the GitHub API, it provides a view of the availability trends in the CVE–PoC ecosystem.

Figure 17 shows the cumulative number of repositories that disappear from the monitoring results over time, grouped by CVE-year bucket. Across all years, the cumulative counts increase steadily throughout the monitoring window. Notably, several buckets exhibit a clear increase around late summer 2024, suggesting a short period of elevated removals. Following this, the cumulative curves continue to grow gradually, reflecting ongoing removals across all CVE-year buckets. Overall, removals are observed across older and newer CVE-years. This indicates that repository disappearance is not bound to a single CVE-year and may reflect broader dynamics of PoC availability on GitHub, such as repository deletion, privatization, or changes in discoverability within GitHub search.

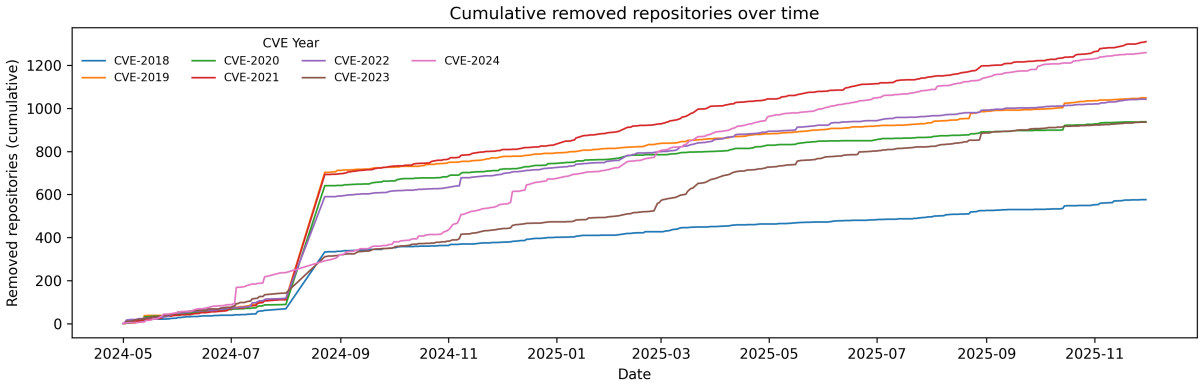

Figure 17: Daily monitored repository deletions over time.

6.7 Social Network Analysis

The GitHub ecosystem with vulnerability-related repositories can be modeled as a social network, where the primary relationship is the link between repository owners and their repositories. Each owner and repository correspond to a node, and edges capture interactions reflecting ownership. While this owner-to-repository graph forms the main analysis, additional relationships, such as fork relations, repository hash and shared CVE identifiers, can be incorporated to examine fork behavior and clustering around similar code or specific vulnerabilities. In this section we define, extract and analyze several networks, generated from the dataset.

Formally, we represent the network of repositories and owners as a bipartite graph G_{owner}, with $G_{\text{owner}} = (V, E_{\text{own}})$, where $V = U \cup R$ is the set of vertices consisting of owners U and repositories R, and $E_{\text{own}} \subseteq U \times R$ contains edges (u, r) from owner $u \in U$ to repository $r \in R$ whenever u owns r.

The GitHub vulnerability ecosystem, in our data, consists of 119,134 nodes and 87,832 edges, representing 31,302 users and 87,832 repositories (Table 16). By definition, each repository is owned by a single user, so for repository nodes the in-degree is fixed at 1. For user nodes, the average out-degree indicates that each user owns about $\frac{|E_{\text{own}}|}{|U|} \approx 2.8$ repositories, but this distribution is highly skewed, with a median of 1 and one user managing up to 2,759 repositories. This skewed out-degree distribution suggests that repository creation is concentrated among a few "influential users," whereas the majority of contributors maintain only a limited number of repositories. These observations motivate further analysis of the network structure and degree distributions to understand the social organization of vulnerability-related activity on GitHub.

Table 16: Summary statistics of the GitHub owner-to-repository network.

Statistic	Value
Total nodes	119,134
Total edges	87,832
Users	31,302
Repositories	87,832
Average repos per user	2.85
Median repos per user	1
Maximum repos per user	2,759

Analysis of repository ownership reveals a concentration among highly active contributors. The top 10 users, representing only 0.03% of all users, own 7.16% of all repositories. This skewed distribution highlights that a small fraction of "influential users" disproportionately contribute to the ecosystem, while the majority of users maintain only a few projects. To quantify the inequality in repository ownership, we compute the Lorenz curve over the distribution of repositories per user and derive the Gini index (Figure 18). The Gini index is $G = 0.577$, indicating an imbalance in ownership and highlighting the disproportionate influence of a small set of highly active repository owners in the ecosystem.

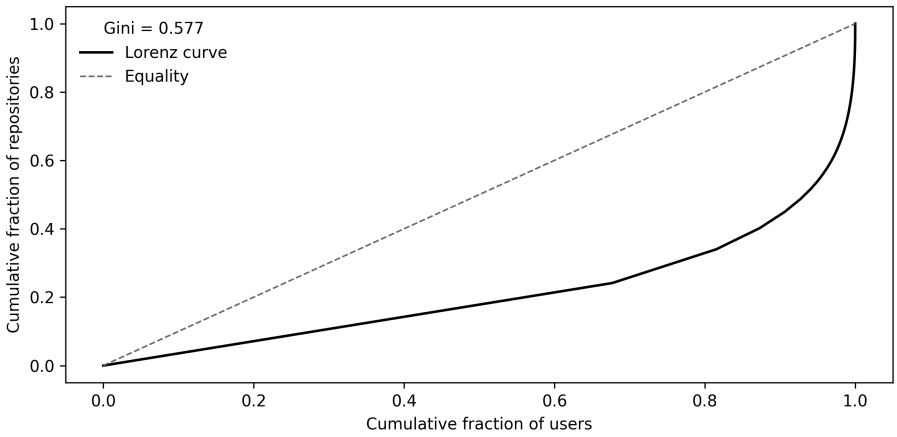

Figure 18: Lorenz curve of repository ownership (repositories per user) with the line of equality.

Before modeling the fork graph, we analyze the completeness of forked repository metadata. Among the 74,687 repositories labeled as forks, 2,627 (3.5%) reference a `parent_repository` that does not appear in our dataset, corresponding to 509 distinct upstream parents. For these repositories, we still instantiate a node for the parent (using its user and repository name) and connect the child to it with a fork edge. However, we do not have any further metadata about these parents (e.g., whether they are forks themselves or repository contents), so these nodes act as boundary vertices in the fork graph: they terminate the observed fork tree and prevent us from reconstructing the possible full upstream fork structure.

In addition to ownership relations, we model fork relations between repositories as a directed graph $G_{\text{fork}} = (R, E_{\text{fork}})$ over the set of repositories R and where E_{fork} is the set of edges between an original repository and its fork. An edge $(r_i, r_j) \in E_{\text{fork}}$ is present whenever r_i is a fork of r_j according to GitHub metadata.

Together, the ownership graph G_{owner} and fork graph G_{fork} define a multi-layer representation of the ecosystem, in which E_{own} captures user repository ownership relations and E_{fork} captures repository fork relations.

For subsequent analysis, we are interested not only in individual fork edges, but in larger groups of repositories connected by series of forks. For this, we consider the undirected variant $G_{\text{fork}}^{\text{und}} = (R, E_{\text{fork}}^{\text{und}})$ obtained by replacing each directed fork edge $(r_i, r_j) \in E_{\text{fork}}$ with an undirected edge $\{r_i, r_j\}$. Two repositories $r, r' \in R$ are fork-reachable if there exists a path between them in $G_{\text{fork}}^{\text{und}}$.

Fork reachability partitions R into connected components, which we interpret as fork communities: sets of repositories that belong to the same fork family. Note that for the rest of the analysis, we exclude the connected components consisting of a single repository, because they do not represent forking behavior.
For this fork reachability we introduce an equivalence relation $\sim_{\text{fork}}$ on R, where

$$r \sim_{\text{fork}} r' \quad \Leftrightarrow \quad \text{there exists a path between } r \text{ and } r' \text{ in } G_{\text{fork}}^{\text{und}}.$$

With these equivalence classes of $\sim_{\text{fork}}$, we describe all connected components of $G_{\text{fork}}^{\text{und}}$. We denote the resulting set of components by

$$\mathcal{C}_{\text{fork}} = R/\sim_{\text{fork}}$$

and refer to each $C \in \mathcal{C}_{\text{fork}}$ as a *fork community*. For a repository $r \in R$, its fork community is

$$C(r) = \{r' \in R \mid r' \sim_{\text{fork}} r\},$$

which consists of all repositories that are fork-reachable from r and so belong to the same fork family.

The defined fork communities capture purely technical fork relationships between repositories. To introduce the social layer for these families, we link fork communities back to owners via ownership relations in E_{own}.
For a fork community $C \in \mathcal{C}_{\text{fork}}$, we define the set of its associated owners as

$$U(C) = \{\, u \in U \mid \exists r \in C : (u, r) \in E_{\text{own}} \,\},$$

resulting in all owners that maintain at least one repository in C. Symmetrically, for an owner $u \in U$ we define the set of fork communities in which u participates as

$$\mathcal{C}_{\text{fork}}(u) = \{\, C \in \mathcal{C}_{\text{fork}} \mid \exists r \in C : (u, r) \in E_{\text{own}} \,\}.$$

The cardinality $|\mathcal{C}_{\text{fork}}(u)|$ measures the activity of an owner in fork families: owners with $|\mathcal{C}_{\text{fork}}(u)| = 1$ contribute to a single repository family, whereas owners with $|\mathcal{C}_{\text{fork}}(u)| > 1$ span across multiple fork families and potentially connect otherwise disjoint families of forks. In addition to the number of fork families per owner, the sizes of the corresponding communities are also informative. For an owner u, the set of fork communities u participates in, captures whether their activity is concentrated in small fork families or in large ones.

Table 17 summarizes statistics for the number of owners per fork family, $|U(C)|$, and for the number of fork families per owner, $|\mathcal{C}_{\text{fork}}(u)|$. We identify a total of $N_f = 6{,}124$ fork families (connected components in the undirected fork graph). The size of families shows a strongly right-skewed distribution: the number of repositories per family has median $\tilde{r} = 4.0$ and mean $\bar{r} = 12.77$, with an interquartile range of $[Q1, Q3] = [2, 10]$, a 95th percentile of 50, and maximum 1081. Similarly, the number of unique owners per family has median $\tilde{o} = 4.0$ and mean $\bar{o} = 12.76$, with $[Q1, Q3] = [2, 10]$ and the same 95th percentile and maximum (50 and 1081, respectively). Together, these results indicate that most fork families are small and involve only a few distinct owners (both typically between 2 and 10), while a small fraction of families span across many repositories and owners. Note that the number of owners counts distinct users, therefore $\bar{o} \leq \bar{r}$, as a single owner may host multiple repositories within the same fork family. The fork families per owner has median $\tilde{f} = 1.0$ and mean $\bar{f} = 2.84$, with $[Q1, Q3] = [1, 2]$, a 95th percentile of 8, and a maximum of 2660. This indicates that most repository owners participate in only one or a few fork families, whereas a small number of owners is associated with an extremely large number of fork families.

Taken together, both the repositories and owners per fork family, as well as the fork families per owner, show strongly right-skewed distributions. These patterns indicate that fork families are typically concentrated around a small set of repositories and owners, while a small number of highly active owners participate in a disproportionately large number of fork families, linking many otherwise separate communities.

Table 17: Base statistics for fork families (connected components).

Metric	Min	Q1	Median	Q3	Mean	P95	Max
Repos per fork family	2	2.0	4.0	10.0	12.77	50.0	1081
Unique owners per fork family $\lvert U(C)\rvert$	1	2.0	4.0	10.0	12.76	50.0	1081
Fork families per owner $\lvert\mathcal{C}_{\text{fork}}(u)\rvert$	1	1.0	1.0	2.0	2.84	8.0	2660

Our analysis has treated fork families as structural objects, focusing on their size and composition. This perspective revealed a strong skew in sizes, with a small number of large fork families and many small ones. However, it does not capture how fork-related activity evolves over time. To add to this view, we introduce a temporal labeling of fork families based on the CVE years associated with the repositories in these families. Each family is annotated with the set of CVE year labels from its repositories. This allows us to examine how fork communities are distributed across CVE years and to provide an alternative view of activity over time.

After CVE year labeling, we observe that fork families are overwhelmingly single year-local: $6{,}097$ families (99.56%) are associated with a single CVE year, while only 27 families (0.44%) span across multiple CVE years. This further suggests that forking lineages are typically organized around a specific vulnerability context, consistent with the intuition that forks primarily propagate PoCs targeting the same CVE (or closely related vulnerabilities within the same vendor/technology).

The extreme skew toward single-year families provides additional evidence that code reuse via forking is largely homogeneous with respect to CVE context. This interpretation is also supported by the example introduced earlier. Figure 8 shows the fork tree of a repository originally named `CVE-2017-10271`. Several forks were renamed to `CVE-2017-10272` while retaining identical contents, illustrating how the same underlying code lineage may be relabeled across closely related CVEs without substantial modification.

Table 18 provides a year-based view of fork-family activity. Consistent with our earlier observations on the increasing number of repositories and forks over time, we observe a general rise in the number of fork families in more recent years. We note that the counts for `CVE-2023` and `CVE-2024` are affected by the time of data collection and therefore represent incomplete years, for which additional fork-family activity is expected to emerge.

Table 18: Distribution of fork families by CVE year.

CVE year	# fork families	Share (%)
`CVE-2016`	187	3.05
`CVE-2017`	426	6.96
`CVE-2018`	500	8.16
`CVE-2019`	790	12.90
`CVE-2020`	846	13.81
`CVE-2021`	1,232	20.11
`CVE-2022`	1,015	16.57
`CVE-2023`	727	11.88
`CVE-2024`	428	6.99
Single-year families	**6,097**	**99.56**
Multi-year families	**27**	**0.44**

7 Discussion and Limitations

This section discusses the main findings of the study and reflects on their broader implications for understanding CVE-related PoC activity on GitHub. In particular, it interprets the observations in repository creation, forking, vulnerability coverage, and temporal dynamics, and relates them to the vulnerability disclosure and security research ecosystem. In addition, it outlines the limitations of the data collection and analysis pipeline that should be considered when interpreting the results.

7.1 Discussion

What is the size and structure of the CVE-related PoC ecosystem, and how much repository content is duplicated via forks or potential re-uploads?
GitHub has become a prominent platform for sharing code, such as proof-of-concept exploit code. The collected data shows a large number of publicly available repositories that reference CVEs. After deduplication, the final dataset consists of 19,988 unique PoC repositories, derived from a total of 87,832 non-unique repositories. From which, only a limited number of repositories correspond to original PoC repositories, while roughly 85% are forks. This imbalance suggests an ecosystem in which a relatively small set of authors publishes original PoC repositories, while the majority primarily engage with this code through forking, whether to adapt or save it.

The analysis of unique repositories, after deduplication, shows a clear trend across CVE publication years. The number of unique PoC repositories increases over time, with increases in years that include widely publicized, high-impact vulnerabilities, such as 2021. These spikes indicate that major vulnerabilities can trigger substantial PoC-repository creation. In contrast, the lower counts for the most recent CVE-years (2023 and 2024) are best explained by the timing of data collection, which was primarily in the first half of 2024. Additional repositories for these years are likely to appear after the observation window. Overall, these patterns suggest an ecosystem with few original authors and many forking users, where PoC-repository creation increases in years marked by high-impact vulnerabilities.

Which implementation languages are the most common?
In terms of implementation, the reported programming languages are dominated by Python, C/C++, Java, and Shell, which aligns with common practices in exploit development and security tooling, where rapid prototyping and broad system support are important. At the same time, a large share of repositories have no language label from GitHub's Linguist tooling, largely due to unchanged forked repositories. This further supports the observation that the ecosystem consists mostly of duplicated PoC-repositories rather than newly created PoC-repositories.

Which vulnerabilities are most commonly targeted by PoCs on GitHub, and which vulnerabilities receive the highest community engagement as reflected by forks?
The distribution of referenced CVE-identifiers further highlights how attention is spread within the ecosystem. A small set of widely publicized, high-impact vulnerabilities dominate GitHub's PoC landscape, with vulnerabilities such as Log4Shell, BlueKeep, SMBGhost, and Zerologon that attracted significant attention. These CVEs are associated with remote code execution and network attack vectors, which makes them both practically relevant and attractive for ex-

ploit development. The statistical comparison between CVEs referenced in original and forked repositories shows that forks amplify a subset of vulnerabilities. These vulnerabilities are on average, more severe, suggesting that community propagation disproportionately favors impactful CVEs rather than representing a random sample of vulnerabilities. In cases where referenced CVE identifiers differ in forked repositories compared to the original, users show behavior such as renaming or clustering around related vulnerabilities. All together, the CVE-level analysis indicates that the ecosystem's "attention" is towards severe, low-complexity, remotely exploitable vulnerabilities, and that forked repositories act as a mechanism for this to propagate.

How does PoC publication timing relate to CVE disclosure?
The timing analysis shows that PoC repository creation clusters around CVE disclosure, with activity peaking shortly after publication, with decay over time. The fact that nearly 40% of repositories are created within 30 days and over half within 100 days indicates that PoC repositories are mainly created during a short post-disclosure window, when patches are being deployed, and awareness is high, whici is followed by a long tail of lower activity. The repositories created before the NVD publication date do not necessarily indicate pre-disclosure exploitation, but rather reflects the structure of the CVE assignment process: many CVEs are first published or referenced by CNAs or vendors before they appear in the NVD database, and some repositories may retroactively adopt CVE identifiers. Overall, these results indicate that PoC repository development is tightly linked to public disclosure timelines, with a short, intense period of activity around publication and a smaller amount of background activity before and long after the CVE is published by the NVD.

What does daily monitoring reveal about repository creation and deletion?
The monitoring results complement the static dataset by revealing the dynamic nature of CVE-related PoC activity on GitHub. By focusing on days with complete monitored coverage, the monitoring view shows that repository availability is relatively stable. The composition changes slightly over time, as in more recent CVE-years, particular in `CVE-2024`, more new repositories are created than for older years. Tracking creations and deletions further highlights that PoC repositories are not static. New repositories primarily target recent CVE-years, whereas removals occur across both old and new CVE-years, reflecting a mixture of deletion, privatization, and changes in search visibility. Overall, the monitoring data describes a continuously changing ecosystem, with growth in the newest CVE-years, and removals across all CVE-years.

How concentrated is PoC activity among users, and what does the fork network reveal about forking communities?
The social network perspective highlights a strongly uneven structure in the GitHub ecosystem. The ownership graph shows that most users own only a single repository, while a small set owns a disproportionately large number of repositories, reflected in a Gini index of 0.577 and the finding that the top 0.03% of users account for 7.16% of all repositories. The fork graph shows a large number of small fork families with median sizes of four, heavily tailed by a few very large fork families with hundreds of repositories. Temporal labeling of these fork families reveals that they are almost always associated with a single CVE year, indicating that fork activity is closely linked to a specific CVE-year rather than spanning multiple years. Together, these observations suggest an ecosystem with a small number of highly active contributors, with replication through forking, concentrated in CVE-years.

Taken together, these findings have several implications for the broader vulnerability disclosure and exploit ecosystem. The concentration of original PoC publication among a small set of users, combined with a rapid surge of forks shortly after disclosure, suggests that exploit code becomes widely available within a short time window, often before patches are fully in place. This suggests that the real-world exploitation risk depends not only on vulnerability severity, but also on how quickly PoCs are shared and reused on platforms like GitHub.

The fact that forks disproportionately propagate high-severity and remotely exploitable vulnerabilities further indicates that community attention focuses on risk rather than mirroring all disclosed vulnerabilities. In practice, this means that defenders cannot assume that all disclosed CVEs receive equal treatment in the wild, as a relatively small subset attracts intense PoC activity and rapid dissemination. For security practitioners, this highlights the value of monitoring PoC activity on platforms like GitHub as a complementary signal to traditional vulnerability metrics, potentially enabling earlier prioritization and patching of vulnerabilities.

For GitHub as a platform, the observed dynamics of repository creation, deletion, privatization, and changes in search visibility suggest that PoC availability is fluid rather than static. Exploit code may appear and possibly spread via forks, but could later become harder to discover. This has implications for both defenders and researchers: single measurements may significantly underestimate past and total exposure, while longitudinal monitoring is necessary to capture the more complete nature of exploit dissemination.

Finally, the uneven ownership and fork-family structure emphasizes GitHub's role as a central hub in PoC dissemination. A small number of highly active users act as initial publishers of PoC repositories, while the broader GitHub community primarily amplifies this content through forking. This hub structure suggests that authors could have a disproportionate impact on the spread of exploit code. For researchers, these dynamics motivate further work on identifying early signals of PoC dissemination, understanding the lifecycle of these repositories, and assessing how platform design choices influence the balance between responsible disclosure, defensive awareness, and misuse.

7.2 Limitations

Interpretation of these results requires care due to several scope and measurement limitations. First, our PoC dataset may not be fully representative of all PoCs available on GitHub. It only includes public repositories that explicitly reference CVE identifiers and were successfully returned by the GitHub API, and therefore excludes private repositories, non-CVE-referenced PoCs, and PoCs hosted on other platforms. In addition, the GitHub search API itself is not fully reliable: queries can be affected by rate limiting and failures. We mitigated this by re-querying each query up to two times at each collection moment, but some repositories may still have been missed. Second, our dataset does not include PoCs for CVEs issued after August 2024, as data collection and review were completed before that point. The study therefore covers PoCs published over a nine-year period (for CVE-IDs issued in 2016–2024), which we consider substantial and hope to be useful for the community. Third, repository metadata including CVE references are self-reported and may be incomplete, inconsistent, or retroactively modified.

Fourth, the monitoring system is affected by the same API constraints, producing occasional incomplete days that required filtering and cleaning. This introduced some uncertainty around short-lived creation and disappearance events. Finally, disappearance of a repository from the monitored search results can reflect several causes, such as deletion, privatization, renaming, or changes in search visibility, which the we cannot distinguish. Taken together, the study provides a detailed view of public available, PoC-repository dynamics on GitHub, but its findings should not be generalized to the entire exploit ecosystem without considering these limitations.

8 Conclusions and Future Work

8.1 Conclusions

Public proof-of-concept exploit code has become an important part of the vulnerability ecosystem, supporting validation, defenses and security research. GitHub hosts large collections of such repositories. In this study we aimed to understand the dynamics of their publication, dissemination, and reuse, by conducting a repository-centric study of CVE-referenced proof-of-concept code on GitHub, motivated by the dual-use nature of public PoCs and the use of these platforms to find exploit code after vulnerabilities have been disclosed.

Our study combined historical snapshots with longitudinal monitoring to capture both the static landscape and its dynamics. After systematic cleaning and deduplication, we analyzed implementation characteristics, CVE coverage, and temporal relationships between disclosure and repository creation. To capture collaboration patterns, we used social network analysis of repository ownership and forking, and introduced the notion of *fork families* to identify related code lineages. The findings show that original PoC publication is concentrated among a small set of users, while forking is the main form of community engagement. Repository creation strongly clusters around the time of vulnerability disclosure, indicating the relation between public vulnerability disclosure and PoC-repository availability. CVE references disproportionately highlight severe, remotely exploitable vulnerabilities, and longitudinal monitoring reveals continuous new additions and removals. The social network perspective highlights skewed ownership, with small and numerous fork families, and forking behavior centered locally per CVE-year.

Taken together, these results provide an empirical view for understanding how PoC repositories appear and propagate on GitHub. By characterizing the structure and dynamics of this ecosystem, this work provides data and observations that can support understanding and discussions on security code on public platforms such as GitHub.

8.2 Future Work

Future work could extend this study in several directions. First, to broaden the scope beyond GitHub by incorporating additional data sources, such as GitLab, Exploit-DB and vendors, to obtain a more complete picture of PoC dissemination. The data collection methodology itself could be further developed into a monitoring framework, with more adaptive handling of rate limits and integration of external archives to better withstand API limitations. For analysis, enriching repositories with labels, for example, by distinguishing repositories as vulnerability scanners, demonstrations, weaponized or malicious exploits. As well as code-level metrics and static-analysis, that would enable more detailed studies of PoC content, quality, and reuse. Finally, extending the deduplication and network model to capture similarity beyond forks and exact hashes, to make it possible to trace how exploit code evolves and propagates across repositories and over time.

A Appendix: Additional Data

Table 19: Programming languages with repository counts.

Language	Count	Language	Count	Language	Count
Unknown	62683	YARA	21	Elixir	2
Python	12286	Rich Text Format	20	Visual Basic .NET	2
C	3469	Jupyter Notebook	17	Dart	2
C++	1665	Visual Basic	10	ASP.NET	2
Java	1617	Kotlin	10	Jinja	2
Shell	1322	HCL	10	XSLT	2
JavaScript	801	Assembly	9	Groovy	2
Go	752	CodeQL	8	EJS	2
HTML	663	Nim	6	Astro	2
Ruby	502	Roff	5	Smali	1
C#	485	PostScript	5	M4	1
PHP	338	VimL	4	DIGITAL Command Language	1
PowerShell	273	TeX	4	GDScript	1
Dockerfile	235	Smarty	4	CMake	1
Objective-C	113	ASP	3	Open Policy Agent	1
Lua	89	Clojure	3	Coq	1
Batchfile	86	VBScript	3	Scheme	1
Rust	67	Crystal	2	Objective-C++	1
ActionScript	31	Logos	2	WebAssembly	1
CSS	31	ApacheConf	2	NSIS	1
Zeek	30	Bro	2	Tcl	1
Swift	30	CoffeeScript	2	Fortran	1
Perl	27	VBA	2	SCSS	1
TypeScript	25	Hack	2	Julia	1
Makefile	21	Scala	2		

References

[BB14] Marco Biazzini and Benoit Baudry. " may the fork be with you": novel metrics to analyze collaboration on github. In *Proceedings of the 5th international workshop on emerging trends in software metrics*, pages 37–43, 2014.

[BGL+09] Joel Brandt, Philip J Guo, Joel Lewenstein, Mira Dontcheva, and Scott R Klemmer. Writing code to prototype, ideate, and discover. *IEEE software*, 26(5):18–24, 2009.

[BHV16] Hudson Borges, Andre Hora, and Marco Tulio Valente. Understanding the factors that impact the popularity of github repositories. In *2016 IEEE international conference on software maintenance and evolution (ICSME)*, pages 334–344. IEEE, 2016.

[BNM21] Guru Bhandari, Amara Naseer, and Leon Moonen. Cvefixes: automated collection of vulnerabilities and their fixes from open-source software. In *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE 2021, page 30–39, New York, NY, USA, 2021. Association for Computing Machinery.

[CCSK24] Siyue Chen, Loek Cleophas, Sandro Schulze, and Jacob Krüger. Use the forks, look! visualizations for exploring fork ecosystems. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 993–1004. IEEE, 2024.

[CS15] Fragkiskos Chatziasimidis and Ioannis Stamelos. Data collection and analysis of github repositories and users. In *2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pages 1–6, 2015.

[Cyb19] Cybersecurity and Infrastructure Security Agency (CISA). CISA microsoft operating systems bluekeep vulnerability. `https://www.cisa.gov/news-events/cybersecurity-advisories/aa19-168a`, 2019. Accessed in January 2026.

[Cyb21] Cybersecurity and Infrastructure Security Agency (CISA). CISA mitigating log4shell and other log4j-related vulnerabilities. `https://www.cisa.gov/news-events/cybersecurity-advisories/aa21-356a`, 2021. Accessed in January 2026.

[DLC+25] Wenjing Dang, Kaixuan Li, Sen Chen, Zhenwei Zhuo, Lyuye Zhang, and Zheli Liu. Real-world usability of vulnerability proof-of-concepts: A comprehensive study. *arXiv preprint arXiv:2510.18448*, 2025.

[DSG21] Bindu Dodiya, Umesh Kumar Singh, and Vivaan Gupta. Trend analysis of the cve classes across cvss metrics. *International Journal of Computer Applications*, 975:8887, 2021.

[DWCH25] Lingyan Ding, Xingya Wang, Zhenyu Chen, and Song Huang. Posvia: Inconsistency analyzer for open-source proof-of-concept reports. *Information and Software Technology*, 188:107868, 2025.

[DYZ+20] Kun Du, Hao Yang, Yubao Zhang, Haixin Duan, Haining Wang, Shuang Hao, Zhou Li, and Min Yang. Understanding promotion-as-a-service on github. In *Proceedings of the 36th Annual Computer Security Applications Conference*, pages 597–610, 2020.

[ENI25] ENISA. Enisa threat landscape 2025. `https://www.enisa.europa.eu/publications/enisa-threat-landscape-2025`, 2025. Accessed in January 2026.

[FNFV23] Elena Fedorchenko, Evgenia Novikova, Andrey Fedorchenko, and Sergei Verevkin. An analytical review of the source code models for exploit analysis. *Information*, 14(9), 2023.

[Git24] GitHub. About github and git, 2024. Accessed in November 2024.

[HBL+19] Sameera Horawalavithana, Abhishek Bhattacharjee, Renhao Liu, Nazim Choudhury, Lawrence O. Hall, and Adriana Iamnitchi. Mentions of security vulnerabilities on reddit, twitter and github. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 200–207, 2019.

[HWRC18] Yan Hu, Shanshan Wang, Yizhi Ren, and Kim-Kwang Raymond Choo. User influence analysis for github developer social networks. *Expert Systems with Applications*, 108:108–118, 2018.

[HYB+24] Hao He, Haoqin Yang, Philipp Burckhardt, Alexandros Kapravelos, Bogdan Vasilescu, and Christian Kästner. 4.5 million (suspected) fake stars in github: A growing spiral of popularity contests, scams, and malware. *arXiv preprint arXiv:2412.13459*, 2024.

[JLH+17] Jing Jiang, David Lo, Jiahuan He, Xin Xia, Pavneet Singh Kochhar, and Li Zhang. Why and how developers fork what from whom in github. *Empirical Software Engineering*, 22(1):547–578, 2017.

[KGB+14] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 92–101, New York, NY, USA, 2014. Association for Computing Machinery.

[LCB22] Triet HM Le, Huaming Chen, and M Ali Babar. A survey on data-driven software vulnerability assessment and prioritization. *ACM Computing Surveys*, 55(5):1–39, 2022.

[LRM14] Antonio Lima, Luca Rossi, and Mirco Musolesi. Coding together at scale: Github as a collaborative social network. In *Proceedings of the international AAAI conference on web and social media*, volume 8, pages 295–304, 2014.

[MV18] Thaís Mombach and Marco Tulio Valente. Github rest api vs ghtorrent vs github archive: A comparative study, 2018.

[MYB19] Vanamala Mounika, Xiaohong Yuan, and Kanishka Bandaru. Analyzing cve database using unsupervised topic modelling. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 72–77, 2019.

[NZ10] Stephan Neuhaus and Thomas Zimmermann. Security trend analysis with cve topic models. In *2010 IEEE 21st International Symposium on Software Reliability Engineering*, pages 111–120, 2010.

[OM22] Ahmet Okutan and Mehdi Mirakhorli. Predicting the severity and exploitability of vulnerability reports using convolutional neural nets. In *Proceedings of the 3rd International Workshop on Engineering and Cybersecurity of Critical Systems*, EnCyCriS '22, page 1–8, New York, NY, USA, 2022. Association for Computing Machinery.

[Ous98] John K Ousterhout. Scripting: Higher level programming for the 21st century. *Computer*, 31(3):23–30, 1998.

[PFNS23] Piotr Przymus, Mikołaj Fejzer, Jakub Narębski, and Krzysztof Stencel. The secret life of cves. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, pages 362–366, 2023.

[Rap24] Rapid7. Metasploit framework, 2024. Accessed in November 2024.

[SCG19] Madeline Schiappa, Graham Chantry, and Ivan Garibay. Cyber security in a complex community: A social media analysis on common vulnerabilities and exposures. In *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 13–20, 2019.

[Sec24] Offensive Security. Exploit database, 2024. Accessed in November 2024.

[SIG24] CVSS SIG. Common vulnerability scoring system sig, 2024. Accessed in November 2024.

[SNL+22] Octavian Suciu, Connor Nelson, Zhuoer Lyu, Tiffany Bao, and Tudor Dumitras. Expected exploitability: Predicting the development of functional vulnerability exploits. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 377–394, Boston, MA, August 2022. USENIX Association.

[SSD15] Carl Sabottke, Octavian Suciu, and Tudor Dumitraș. Vulnerability disclosure in the age of social media: Exploiting twitter for predicting {Real-World} exploits. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 1041–1056, 2015.

[SSM+20] Prasha Shrestha, Arun Sathanur, Suraj Maharjan, Emily Saldanha, Dustin Arendt, and Svitlana Volkova. Multiple social platforms reveal actionable signals for software vulnerability awareness: A study of github, twitter and reddit. *Plos one*, 15(3):e0230250, 2020.

[TBLJ13] Ferdian Thung, Tegawende F Bissyande, David Lo, and Lingxiao Jiang. Network structure of social coding in github. In *2013 17th European conference on software maintenance and reengineering*, pages 323–326. IEEE, 2013.

[The22] Robin The. Analysis of exploit code published on github, 2022. Supervisors: Olga Gadyatskaya & Soufian El Yadmani.

[WDZ+25] Lingxiao Wang, Wenjing Dang, Mengyao Zhao, Yue Wang, Xianzong Wu, and Sen Chen. Aligning core aspects: Improving vulnerability proof-of-concepts via cross-source insights. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, pages 1774–1777, 2025.

[YTG22] Soufian El Yadmani, Robin The, and Olga Gadyatskaya. Beyond the surface: Investigating malicious cve proof of concept exploits on github. *arXiv preprint arXiv:2210.08374*, 2022.

[YTG25] Soufian El Yadmani, Robin The, and Olga Gadyatskaya. {SecurePoC}: A helping hand to identify malicious {CVE} proof of concept exploits in {GitHub}. In *19th USENIX WOOT Conference on Offensive Technologies (WOOT 25)*, pages 263–282, 2025.

[YYWW14] Yue Yu, Gang Yin, Huaimin Wang, and Tao Wang. Exploring the patterns of social behavior in github. In *Proceedings of the 1st international workshop on crowd-based software development methods and technologies*, pages 31–36, 2014.

[ZZM14] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. Patterns of folder use and project popularity: a case study of github repositories. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, New York, NY, USA, 2014. Association for Computing Machinery.