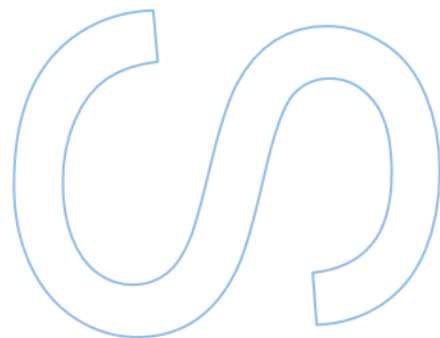
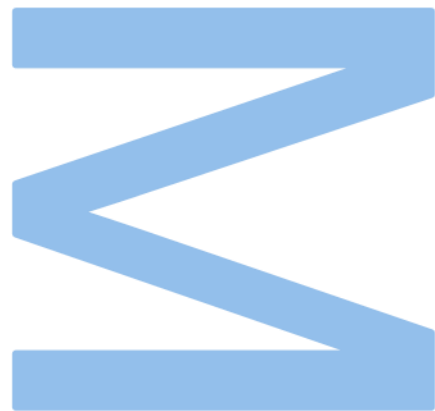


# Effort Estimation for Security Fixes: Towards Better Alert Prioritization

João Paulo Santiago

Master in Information Security  
Department of Computer Science  
Faculty of Sciences of the University of Porto  
2025



# Effort Estimation for Security Fixes: Towards Better Alert Prioritization

**João Paulo Santiago**

Dissertation carried out as part of the Master in Information  
Security

Department of Computer Science  
2025

**Supervisor**

Rui Maranhão Abreu, Full Professor, FEUP

**External Host Supervisor**

Sofia Reis, Invited Assistant Professor, FEUP

# Acknowledgements

I would like to express my sincere gratitude to all those who have supported me throughout the course of this research and the writing of this thesis.

To Rui and Sofia for their invaluable input and contributions that made this work possible.

To my family, for their unwavering support throughout all my academic journey.

And to my girlfriend, for her words of comfort and encouragement that still continue to push me through thick and thin.

# Resumo

À medida que os ecossistemas de software se tornam mais complexos, a correção rápida e eficaz de vulnerabilidades de segurança assume um papel central na manutenção de software orientada para a cibersegurança. Apesar da investigação extensa sobre detecção de vulnerabilidades e implementação de patches, os actuais modelos de gestão de vulnerabilidades tendem a negligenciar o esforço de desenvolvimento necessário para implementar correções seguras e fiáveis. Compreender este esforço é crucial, pois influencia diretamente a qualidade do patch e a resiliência global dos sistemas de software.

Esta dissertação apresenta uma investigação empírica sobre as características estruturais de patches de segurança em Python, desenvolvidos no contexto de software open source, analisando se as mudanças de código por eles introduzidas se correlacionam com a dificuldade prática da sua implementação. Para quantificar a complexidade dos patches, é proposto um Structural Churn Score, que integra múltiplas métricas de dimensão e complexidade, incluindo o número de linhas modificadas ( $\Delta\text{LOC}$ ), número de ficheiros afectados ( $\Delta\text{Files}$ ), complexidade ciclomática ( $\Delta\text{CC}$ ) e alterações na estrutura de controlo ( $\Delta\text{Blocks}$ ,  $\Delta\text{Edges}$ ,  $\Delta\text{Paths}$ ). Este score funciona como um indicador quantitativo do esforço cognitivo e técnico associado às modificações de segurança no código.

A análise efetuada demonstra que os patches de segurança em projetos Python tendem a introduzir um crescimento estrutural localizado, com a maioria das alterações a permanecer relativamente circunscrita. Contudo, o Structural Churn Score evidencia que o aumento da complexidade cognitiva e as mudanças na lógica de controlo são os principais factores que influenciam o esforço de correcção. Estes resultados sublinham a importância de considerar a evolução estrutural do código ao avaliar as exigências operacionais da remediação de vulnerabilidades, contribuindo assim para práticas de cibersegurança mais eficazes.

Palavras-chave: Structural Churn Score, Métricas de Software, Complexidade, Esforço

# Abstract

As modern software ecosystems increase in complexity, the timely and effective remediation of security vulnerabilities has become a central challenge in cybersecurity-oriented software maintenance. Despite extensive research on vulnerability discovery and patch deployment, existing vulnerability management frameworks frequently overlook the development effort required to implement secure and reliable fixes. Understanding this effort is critical, since it influences patch quality, maintainability, and the overall resilience of software systems.

This thesis presents an empirical investigation into the structural characteristics of Python security patches within open-source software, examining whether their code changes correlate with the practical difficulty of implementing these patches. To quantify patch complexity, we introduce a composite Structural Churn Score that integrates multiple size and complexity metrics, including the number of modified lines ( $\Delta\text{LOC}$ ), number of affected files ( $\Delta\text{Files}$ ), cyclomatic complexity ( $\Delta\text{CC}$ ), and changes to control-flow structure ( $\Delta\text{Blocks}$ ,  $\Delta\text{Edges}$ ,  $\Delta\text{Paths}$ ). This score serves as a quantitative proxy for estimating the cognitive and technical effort associated with security-relevant code modifications.

Our analysis demonstrates that security patches in Python projects typically introduce localized structural growth, with most modifications remaining relatively contained in scope. Nevertheless, the Structural Churn Score reveals that increases in cognitive complexity and changes in control-flow logic are the predominant factors influencing patching effort. These findings highlight the importance of considering structural code evolution when assessing the operational demands of vulnerability remediation, ultimately contributing to more effective cybersecurity practices.

Keywords: Structural Churn Score, Software Metrics, Complexity, Effort

# Table of Contents

|                                                                                                                                                                               |     |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| List of Tables .....                                                                                                                                                          | vi  |
| List of Figures.....                                                                                                                                                          | vii |
| 1. Introduction .....                                                                                                                                                         | 1   |
| 1.1. Context.....                                                                                                                                                             | 1   |
| 1.2. Motivation.....                                                                                                                                                          | 1   |
| 1.3. Problem Statement .....                                                                                                                                                  | 2   |
| 1.4. Hypothesis and Research Questions .....                                                                                                                                  | 2   |
| 1.5. Contributions .....                                                                                                                                                      | 3   |
| 1.6. Thesis Structure .....                                                                                                                                                   | 3   |
| 2. State of the Art .....                                                                                                                                                     | 4   |
| 2.1. Vulnerability Prioritization .....                                                                                                                                       | 4   |
| 2.2. Software Metrics .....                                                                                                                                                   | 6   |
| 2.3. Other Metrics .....                                                                                                                                                      | 7   |
| 2.4. Vulnerability Fix Effort .....                                                                                                                                           | 7   |
| 3. Methodology .....                                                                                                                                                          | 9   |
| 3.1. Dataset.....                                                                                                                                                             | 9   |
| 3.2. Software Metrics .....                                                                                                                                                   | 11  |
| 3.2.1. Software Size Metrics .....                                                                                                                                            | 12  |
| 3.2.1.1. Modified Files .....                                                                                                                                                 | 13  |
| 3.2.1.2. Modified Lines of Code .....                                                                                                                                         | 13  |
| 3.2.2. Software Complexity Metrics .....                                                                                                                                      | 14  |
| 3.2.2.1. Cyclomatic Complexity .....                                                                                                                                          | 14  |
| 3.2.2.2. Execution Path Metrics .....                                                                                                                                         | 17  |
| 3.3. Structural Churn Score .....                                                                                                                                             | 19  |
| 3.4. Running the Script .....                                                                                                                                                 | 20  |
| 4. Results.....                                                                                                                                                               | 21  |
| 4.1. What kinds of structural changes are typical in Python security patches? .....                                                                                           | 21  |
| 4.2. Do security patches tend to increase or decrease control-flow complexity? .....                                                                                          | 22  |
| 4.3. Are larger structural changes ( $\Delta$ Blocks, $\Delta$ Edges, $\Delta$ Files) associated with increased control-flow complexity ( $\Delta$ CC, $\Delta$ Paths)? ..... | 23  |
| 5. Discussion and Implications.....                                                                                                                                           | 30  |
| 5.1. Effort Beyond Size: Revisiting the Metrics .....                                                                                                                         | 30  |
| 5.2. Prioritization with Effort-Aware Triage .....                                                                                                                            | 30  |
| 5.3. Quality Gates for Automated Patches .....                                                                                                                                | 31  |
| 5.4. Language-Agnostic Expansion.....                                                                                                                                         | 31  |

|                                              |    |
|----------------------------------------------|----|
| 5.5. Fine-Tuning Metric Weighting.....       | 31 |
| 5.6. Practical Caveats and Limitations ..... | 32 |
| 5.7. Summary.....                            | 33 |
| 6. Conclusion.....                           | 34 |
|                                              |    |
| Bibliography .....                           | 35 |
| .1. Dataset.....                             | 40 |
| .2. Results .....                            | 42 |

## List of Tables

|                                                                                           |    |
|-------------------------------------------------------------------------------------------|----|
| 3.1. Descriptive statistics of the Dataset .....                                          | 10 |
| 3.2. Software Size Metrics and Software Complexity Metrics .....                          | 12 |
| 3.3. Software Size Metrics Description .....                                              | 12 |
| 3.4. Software Complexity Metrics Description .....                                        | 14 |
| 4.1. Descriptive statistics of structural changes in Python security patches .....        | 21 |
| 4.2. Directional changes in control-flow metrics across 100 Python security patches ..... | 22 |
| 5.1. Test cases were comments make up a big part of the LOC .....                         | 32 |

# List of Figures

|                                                                           |    |
|---------------------------------------------------------------------------|----|
| 3.1. Distribution of CWEs across the analyzed security patches. ....      | 11 |
| 4.1. Scatter Matrix for measured Metrics .....                            | 25 |
| 4.2. Correlation Heatmap between Software Metrics .....                   | 26 |
| 4.3. Cyclomatic Complexity by File Count .....                            | 27 |
| 4.4. Blocks by File Count.....                                            | 27 |
| 4.5. Edges by File Count.....                                             | 27 |
| 4.6. Paths by File Count.....                                             | 27 |
| 4.7. Correlation between Structural Churn Score and Software Metrics..... | 29 |



# 1. Introduction

## 1.1. Context

In recent years, cyber security became a major part of software development cycles. Automated tools such as Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and Software Composition Analysis (SCA) are now present in most Continuous Integration and Continuous Delivery (CI/CD) pipelines across the industry, providing developers with updated information regarding their projects security issues.

Security auditors play a vital role in helping developers focus on which vulnerabilities to fix, since the average developer lacks the training necessary regarding cybersecurity topics. By analyzing the many results generated by various security tools, filtering out false positives and elaborating actionable mitigation plans for developers to follow. This workflow is still mainly manual and time consuming. This process is repeated for every project security auditors must oversee, thus increasing the workload of this professionals exponentially [1].

## 1.2. Motivation

In agile DevSecOps environments, rapid software development can outpace remediation efforts, leading to an increase in technical [2] and security debt [3]. All of these factors can culminate in a workload overload that affects both developers as well as the security auditors tasked with helping them, ultimately decreasing the quality and security of the produced software.

A study on the life of software vulnerabilities found that half of the 3,645 vulnerabilities considered remained unfixed for 511 days in the software analyzed [4], a far longer period than what should be acceptable for some class of vulnerabilities [5][6]. Many factors can contribute to this delay in remediation, one of them being the lack of clear guidance on what to fix first.

Conventional prioritization frameworks, such as those based on the Common Vulnerability Scoring System (CVSS), are solely based on exploit severity [7], leaving both developers and security auditors without other clear prioritization criteria. This limitation can lead to inefficient resource allocation and unaddressed vulnerabilities that may be low-severity but trivial to fix.

From professional experience, some developers will also offer some resistance when asked to fix security vulnerabilities due to their already existing backlog of work, claiming that additional workload will delay the release of new features.

### 1.3. Problem Statement

Security patching remains a labor-intensive task, especially when developers must assess and fix vulnerabilities under time pressure. Existing prioritization approaches often focus on vulnerability severity or exploitability, but overlook a key practical factor: the *effort required to implement a secure patch*.

In this work, we focus on *structural patch effort*, that is, the magnitude of code changes required to fix a vulnerability, measured through *static, structural and control-flow metrics*. Unlike concepts such as cognitive effort or maintainability, which often depend on developer experience, readability, or subjective judgment, our definition of effort is structural. We do not attempt to model how a developer feels about a patch, how easy it is to understand, or how it affects long-term code health, nor are we interested in measuring how much time it takes to code a security patch [8]. Instead, we focus on observable, quantifiable changes to the code's structure and control flow, such as the number of modified lines, files, branches, or execution paths. It captures the syntactic and structural disruption introduced by a patch, which we hypothesize contributes to the reasoning and validation burden.

To operationalize this idea, we analyze 100 Python security patches from open-source software projects, extracting metrics such as the number of modified lines ( $\Delta\text{LOCs}$ ), number of modified files ( $\Delta\text{Files}$ ), cyclomatic complexity ( $\Delta\text{CC}$ ), and changes in control-flow structure ( $\Delta\text{Blocks}$ ,  $\Delta\text{Edges}$ ,  $\Delta\text{Paths}$ ). We combine these into a **Structural Churn Score** that serves as a quantitative proxy for patch complexity. Our goal is to establish a baseline for structural patch effort that can inform future prioritization strategies, tooling design, and vulnerability triage workflows.

### 1.4. Hypothesis and Research Questions

A key question in secure software engineering is whether the *size and structure* of a patch, such as how many files it touches or how much it alters control flow, affect the effort required to implement it. Alternatively, effort may depend more on the logical intricacy of the change rather than its structural footprint.

To investigate the relationship between structural change and perceived difficulty, we pose the following research questions:

- **RQ1: What kinds of structural changes are typical in Python security patches?**  
Establishes a quantitative baseline for scope and control-flow changes.
- **RQ2: Do security patches tend to increase or decrease control-flow complexity?** Tests the common assumption that patches harden logic by adding checks or branches.
- **RQ3: Are larger structural changes ( $\Delta$ blocks,  $\Delta$ edges,  $\Delta$ files) associated with increased control-flow complexity (CC,  $\Delta$ paths)?** Explores whether broader code changes correlate with deeper reasoning effort.

## 1.5. Contributions

The results of this work show that most patches are relatively small and localized, although there is a substantial variability across the dataset. They also show that most Python security patches introduce structural growth and complexity, and that the difficulty of a security patch is more closely related to the cognitive complexity of the patch than to the structural changes it introduces.

It is important to note that not all high-severity vulnerabilities require the same effort to fix; some may require a complete overhaul of the program logic and/or architecture while others may be resolved with minimal code changes. Being able to distinguish between these cases may be the key to a more effective vulnerability management, particularly in systems with numerous vulnerabilities present.

## 1.6. Thesis Structure

This thesis is organized in seven sections. The first section contains the introduction to the work. The second section reviews the current state of the art. The third section details the methodology followed throughout the work. The fourth section presents and analyses the obtained results. The fifth section the results are discussed and future work is proposed. Finally, the sixth section concludes this work.

## 2. State of the Art

During this work, three main topics were the focus of our research: prioritization strategies for vulnerability fixing, software metrics and vulnerability fix effort.

Unfortunately, there are not many works out there that try to quantify the amount of work a developer has when it comes to coding. Some works have measured the the developers' contribution on source code quality software metrics [9] while most other works only measure the quality of the source code through the use of software and complexity metrics.

### 2.1. Vulnerability Prioritization

Security is becoming a major focus on software development in recent years through the adoption of the DevSecOps movement [10]. As such, part of this research focused on understanding what strategies were being implemented to deal with vulnerabilities during software development, mainly how different vulnerabilities were prioritized.

A vulnerability is a weakness in the security of the computer system that can allow harm to occur [11].

In 2022, a survey was conducted with the aim of studying various approaches for post-processing of static analysis alarms [12], following up on a previous survey conducted in 2016 [13], where the authors review several strategies for processing alarms generated by static analysis tools, where one of the stated goals for these approaches is to reduce manual inspection effort of the alarms generated.

The approaches reviewed were divided into six different categories: clustering, ranking, pruning, automated false positive elimination, combination of static and dynamic analyses, and simplification of manual inspection, with the main goal of these approaches being to reduce the number of false positives generated by static analysis tools.

Even though the end goal of these approaches is to reduce the workload of developers and security authors that will ultimately be responsible for evaluating the alarms generated, only two of the categories mention the effort needed from these parties.

The clustering of alarms will inherently reduce the overall inspection effort, since alarms in a group are similar or closely correlated, meaning the inspection can be done group-wise and not individually.

The simplification of manual inspection category focuses on approaches that aim to support users during the inspection process. One of the subcategories, alarms-relevant queries, where the root causes of alarms are identified, and queries specific to the causes are presented to the user are generated, thus reducing the manual inspection effort by avoiding redundant effort in locating the cause points and constraints of each alarm.

Another sub category where the user's effort is shown to be reduced when fixing the alarms is the Automated Repair subcategory. As the name implies, the aim of the approaches in this subcategory is the automated repair of alarms. In this case, the effort is reduced for the user because the user must only accept or reject the automated fixes generated.

In none of the categories identified is the effort required by the developers to fix the security alarms being taken into account, with the exception of the Automated Repair subcategory. Even in this subcategory, the effort required is not measured and is simply reduced to a binary choice for the user. This measurement is something that we intend to do in this work.

Other research has proposed prioritization methods focus primarily on the severity of the vulnerabilities found.

In *A Hybrid Approach for Evaluation and Prioritization of Software Vulnerabilities* [14], a prioritization method using the fuzzy Best Worst Method to determine the severity of a given software vulnerability was proposed, followed by a two-way analysis to integrate the opinion of the decision makers, also based on the severity of the vulnerabilities.

In no part of this study is the effort required by developers to fix said software vulnerabilities being considered for the prioritization of the same.

In another study, the description of the vulnerability is used to infer its severity [15][16], using word embeddings and convolutional neural networks (CNN). Again, only severity is inferred from the description, leaving the developers and security auditors with no information on the effort required to fix the vulnerabilities.

Others have proposed prioritization frameworks where time metrics, the total number of

vulnerabilities and the time it took for them to be remediated, TVR (time to vulnerability remediation) where also used [17], in conjunction with the severity level of each vulnerability.

Even in such studies, where the time to remediate is considered, the effort need to remediate is not calculated and can only be guessed based on this time.

In most of these studies, a vulnerability's severity is quantified using the Common Vulnerability Scoring System (CVSS), which presents a numerical score ranging from 0 to 10 [18].

To our knowledge, no work has proposed a prioritization framework based on the effort required to code the fix for a given vulnerability, a gap that we intend to bridge with this work by proposing this new effort score.

## 2.2. Software Metrics

Software Metrics provide a measurement for the software and the process of software production, giving quantitative values to the production process [19]. Three types of software metrics are identified in these work, procedure metrics, project metrics and product metrics.

This work also explores software complexity metrics. Software complexity metrics measure the cost consumed by developing, maintaining, and usage of the software, with complexity being the main factor that can lead to software defects.

These type of metrics are of importance since they can help people maintain projects, select the most appropriate program given its functions, and, more importantly for this work, can help evaluate the workload of programming and cost of development.

Work has been done before where software metrics were used to quantify the contribution in source code made by developers [9]. By computing each of the metrics chosen for a given code, the evolution of a given metric is obtained by comparing it in each commit with its antecessor, and understanding if that difference had affected positively, negatively, or did not influence the code maintainability, testability, reusability, and understandability.

Although the developers' contribution can be quantified, the workload these developers expend to make these contributions remains unknown.

Previous work was also done to measure the software evolution process, comparing the complexity of sequential builds, and a new metric is calculated, code churn [20].

While code churn seems to be a valuable early indicator of software faults, which can be very useful for developers to structure an adequate test plan, it remains as just another software quality indicator, not taking into account nor giving any information about the effort required to fix the software faults it might indicate.

## 2.3. Other Metrics

Other types of metrics have also been proposed in the past.

In *A Suite of Cognitive Complexity Metrics* [21], the authors propose a new set of metrics that use the cognitive aspect of the code in terms of cognitive weight for object-oriented languages. In total, 10 new metrics were proposed, Method Complexity(MC), Message complexity (Coupling Weight for a Class (CWC)), Attribute Complexity(AC), Weighted Class Complexity(WCC), Code Complexity (Inheritance), Average Method Complexity(AMC), Average Method Complexity per Class(AMCC), Average Class Complexity(ACC), Average Coupling Factor(ACF), and Average Attributes per Class(AAC). All of these metrics measure the complexity of a property of an object-oriented programming language, such as a class or a method.

In this thesis, we aim to analyze code written in Python, which supports object-oriented programming, but also propose a more generic formula for the effort score, so it can be more easily exported to other programming languages, which may not be object-oriented. Thus, this set of metrics will not be currently considered, but can be interesting to explore in future works.

## 2.4. Vulnerability Fix Effort

In 2015, a study was conducted to determine which factors impacted the effort required to fix security vulnerabilities [8], identifying 65 contributing factors for vulnerability fix time. These factors were grouped in 8 categories: vulnerabilities characteristics, software structure, diversity of the used technology, communication and collaboration smoothness, availability and quality of the information and documentation, expertise and knowledge of developers, efficiency of the static analysis tool, and other.

In this study, the vulnerability fix effort is measured as the time needed to fix the vulnerability, which differs from the effort we are trying to measure, which is more related to quantifiable changes to the code's structure and control flow.

A human study was made to have a systematic understanding of patch maintainability and understandability, given the recent interest in automatic patch generation [22]. In this study, 150 participants were involved, and 32 real world defects and 40 different patches were used. The participants were asked to demonstrate their understanding of certain aspects of the code, like control-flow and maintainability.

This study concluded that machine-generated patches are slightly less maintainable than human-written ones, a trend that is reversed once the proposed system to augment these patches with synthesized, human-readable documentation is in place.

How patch types affect maintainability was also one of the questions explored in this work. To answer this question, two metrics were used, accuracy and effort. Again, effort is being defined as a temporal metric, the number of minutes that the participants took to understand the patch, which differs from our definition. The authors found that the participants take less time to understand the machine generated patches augmented with documentation versus human-written patches.

To summarize, the content and quality of the comments is a major factor when it comes to improve patch maintainability, lowering the time it takes for developers to understand the patches.

### 3. Methodology

At the outset of this work, we aimed to establish a quantitative baseline for the structural characteristics of security patches. Our goal was to define an effort baseline that captures the typical scope and magnitude of code modifications, and to understand whether security patches tend to harden program logic by adding checks or simplify it by removing redundant or unsafe paths.

To operationalize these goals, we formulated the following research questions:

- **RQ1: What kind of structural changes are typical in Python security patches?**  
To provide a quantitative baseline of a security patch, by establishing a structural profile of typical security patches, which can be useful for tooling and prioritization.
- **RQ2: Do security patches tend to increase or decrease control-flow complexity?** Determining whether security patches generally harden logic or simplify it.
- **RQ3: Are larger structural changes ( $\Delta$ blocks,  $\Delta$ edges,  $\Delta$ files) associated with increased control-flow complexity (CC,  $\Delta$ paths)?** Identify if broader changes tend to make the control flow harder to follow, or if control-flow needs to be tracked independently of the size of the change.

The following sections describe the metrics used to capture these dimensions and explain how each was computed and analyzed.

#### 3.1. Dataset

For this study, we selected a set of security patch use cases from **Open Source Vulnerabilities (OSV)**, a distributed vulnerability database designed to address many of the challenges in identifying, tracking, and analyzing vulnerabilities in open source software [23]. OSV aggregates standardized vulnerability metadata across multiple ecosystems, providing structures links between CVEs, affected packages, and their corresponding commits.

To ensure the quality and consistency of our analysis, each selected vulnerability had to satisfy the following criteria:

- The patch must belong to a project in the PyPI ecosystem;

- A **fix commit** must be explicitly referenced and available in the OSV record;
- The commit must have a **single parent**, since the `Pydriller` library cannot reliably reconstruct the source code of files when the commit is a merge, making it impossible to accurately extract the structural information.

At the time of writing (September 2025), OSV reported roughly 16,000 vulnerabilities affecting the PyPI ecosystem. From these, we selected 100 distinct security patches disclosed between June 2024 and August 2025 that satisfy the above criteria. Each patch was retrieved and analyzed together with its corresponding OSV entry, including meta-data such as the CWE identifier, CVSS v3.0/v4.0 severity score, and the fix commit hash.

The resulting dataset captures a diverse range of patch types and severities, from *LOW* to *HIGH*, with an average CVSS score of 6.66, a minimum of 1.7, and a maximum of 10. Table 3.1 summarizes the descriptive statistics of severity scores across all analyzed security patches, showing a balanced distribution around medium-to-high severity levels.

| Metric  | Severity |
|---------|----------|
| Mean    | 6.66     |
| Std Dev | 2.15     |
| Min     | 1.70     |
| 25%     | 5.3      |
| Median  | 6.9      |
| 75%     | 8.25     |
| Max     | 10.00    |

Table 3.1: Descriptive statistics of the Dataset

The dataset exhibits a diverse range of vulnerability categories. The most frequently observed CWE is CWE-22 (Path Traversal), appearing in 15 patches. Other prevalent categories include CWE-502 (Deserialization of Untrusted Data) with 9 occurrences, CWE-770 (Resource Management Errors / Allocation Without Limits or Throttling) with 7, CWE-79 (Cross-Site Scripting) with 5, and CWE-1333 (Inefficient Regular Expression Complexity), CWE-20 (Improper Input Validation), and CWE-89 (SQL Injection), each with 4 occurrences. The remaining CWEs appear less frequently, typically between 1 to 3 times.

Overall, the dataset covers a wide spectrum of vulnerability classes, including input validation flaws, authentication and authorization issues, cryptographic weaknesses, injection vulnerabilities, denial-of-service risks, and configuration issues. This diversity ensures a representative coverage of common security weaknesses encountered in real-world Python projects.

Figure 3.1 presents the distribution of CWEs across the dataset, while the complete list of CWEs and their frequencies is provided in the appendix.

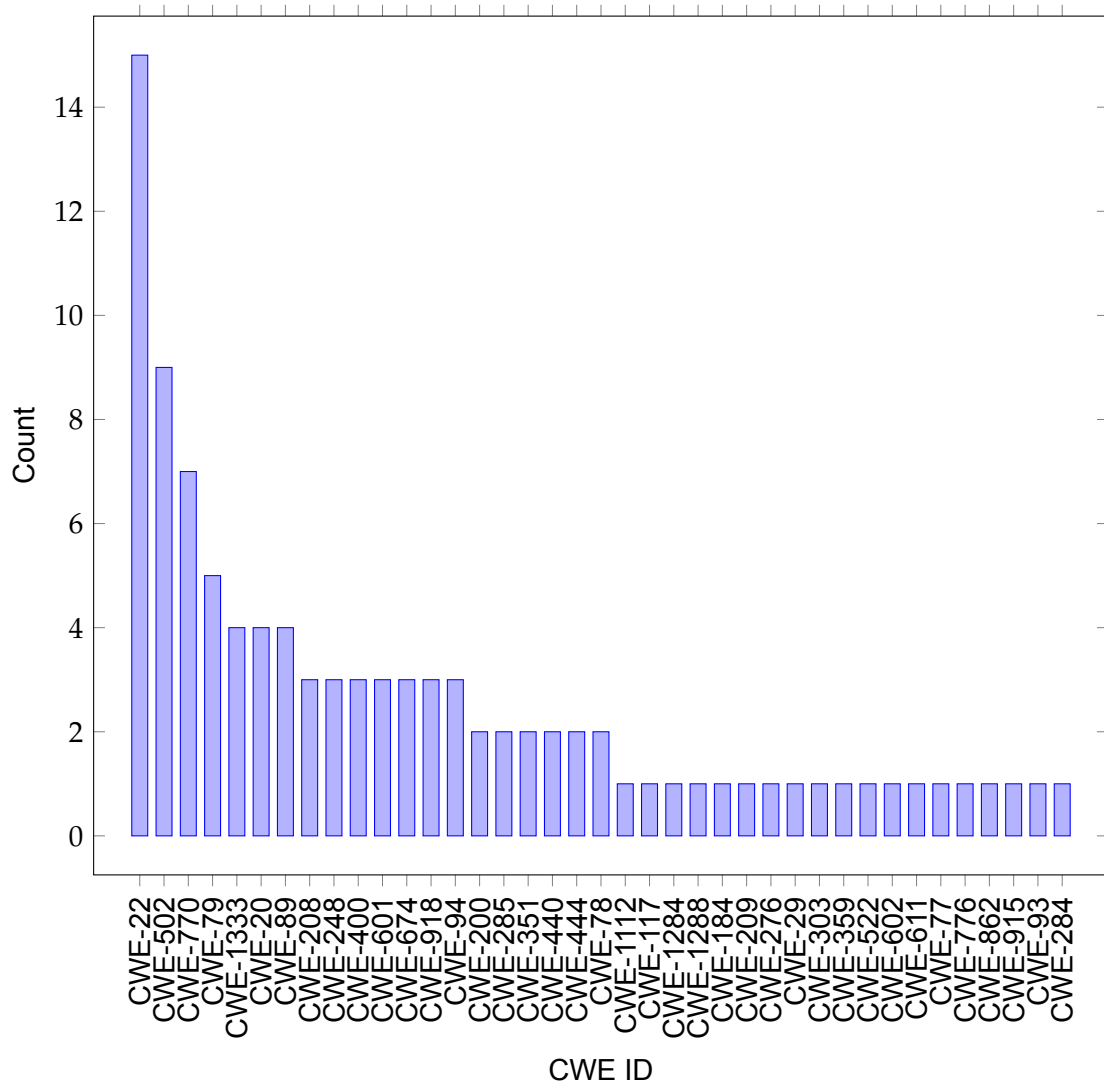


Figure 3.1: Distribution of CWEs across the analyzed security patches.

## 3.2. Software Metrics

In the quantitative analysis of software systems, a range of metrics is employed to assess size, structural complexity, and control-flow characteristics. These metrics are essential for understanding maintainability, testability, reliability, and the effort needed for further development or refactoring. The following metrics are central to the current work: number of lines of code, number of files, cyclomatic complexity, number of blocks, number of edges, and number of paths. This metrics can be divided into 2 different groups: Software Size Metrics and Software Complexity Metrics.

| Aspect     | Software Size Metrics                                                           | Software Complexity Metrics                                                                |
|------------|---------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| Definition | Measure the physical size of software.                                          | Measure the intricacy or difficulty of understanding, maintaining, or modifying software.  |
| Focus      | Quantitative attributes such as lines of code, number of functions, or modules. | Structural or cognitive attributes such as decision paths, control flow, or nesting depth. |
| Metrics    | Lines of Code (LOC), Number of Files                                            | Cyclomatic Complexity, Number of Blocks, Number of Edges and Number of Paths               |

Table 3.2: Software Size Metrics and Software Complexity Metrics

| Metric         | Description                                                                                                      |
|----------------|------------------------------------------------------------------------------------------------------------------|
| Lines of Code  | The number of lines of code modified with the security patch. Accounts for both added and removed lines of code. |
| Modified Files | The number of files modified with the security patch.                                                            |

Table 3.3: Software Size Metrics Description

### 3.2.1 Software Size Metrics

When it comes to software size metrics, two were selected for this work: the number of different files impacted by the fix and the total number of lines of code modified between the fixed and vulnerable version of the code.

The total number of files modified will indicate the breadth of the patch. The more files are modified during the security patch, the more components it will affect, and the higher the coordination effort required by the developers.

As for the total number of lines of code modified by the commit is computed by adding the number of lines added and the total number of lines removed with the fix, while assuming the workload of adding and deleting code lines have the same weight.

With this metric, we can measure the raw size of the code change brought about by the fix and a baseline for the effort required to code the security patch, more lines modified means more lines for the developer to read, write, interpret and test.

To retrieve the values of these metrics, the tool `pydriller` has chosen. `PyDriller` is a Python framework that helps developers in mining software repositories [24]. With this tool we are able to extract information from any Git repository.

Knowing the hash of the commit related to the fix, we can use `pydriller` to fetch all the information about this specific commit. To do this, we use the `Repository` class, specifying

the path to our locally cloned repository and the hash of the commit of the fix, and using the `traverse_commits()` function to return the selected commits.

```
1 # Get single commit info, needs local repo and commit hash
for commit in Repository(repo_path, single=commit_hash).traverse_commits():
```

Having the commit object now populated, getting the information needed becomes trivial, as they are stored in different properties. The `modified_files_fix` property is being used to store this value and is later passed to the result output.

### 3.2.1.1 Modified Files

Within the commit, first, there is the need to identify the documents relevant to the fixing of the vulnerability within the commit.

By retrieving the list of modified files in the commit, stored in the `modified_files` property of the commit object, we traverse this list, analyzing each individual file, filtering out any non-python file, and any file that contains the word "test" in it.

```
# Get info on modified files after commit
2 modified_files_info = {}
for file in commit.modified_files:
4     extension = os.path.splitext(file.filename)[1]
    name = os.path.splitext(file.filename)[0]
6     if extension == ".py" and "test" not in name:
```

Counting the number of files that pass this filter will give us the number of related files to the fix in the commit.

### 3.2.1.2 Modified Lines of Code

While iterating through the list of modified files, the file object will contain the commit information for each individual file, which can be accessed using `pydriller`'s properties.

```
modified_files_info[file.filename] = {}
2 modified_files_info[file.filename]['change_type'] = file.change_type.name
modified_files_info[file.filename]['cyclomatic_complexity'] = file.complexity
4 modified_files_info[file.filename]['added_lines'] = file.added_lines
modified_files_info[file.filename]['deleted_lines'] = file.deleted_lines
```

```

6 modified_files_info[file.filename]['net_lines'] = file.added_lines - file.
    deleted_lines
modified_files_info[file.filename]['total_lines'] = file.added_lines + file.
    deleted_lines

```

Although we could use `pydriller`'s properties to retrieve the number of modified lines of code in the commit, this would be inaccurate, as files that might not be related to the fix are taken into account.

```

1 git_info[repo_name]['insertions'] = commit.insertions
  git_info[repo_name]['deletions'] = commit.deletions
3 git_info[repo_name]['total_lines'] = commit.lines

```

### 3.2.2 Software Complexity Metrics

The next set of metrics selected for this study are related to code complexity. The assumption is made that the more complex the vulnerable code, the more effort required to fix it.

| Metric                    | Description                                                                |
|---------------------------|----------------------------------------------------------------------------|
| Cyclomatic Complexity     | Measures the number of nested conditions within the code.                  |
| Number of Blocks          | Sequence of instructions without branches.                                 |
| Number of Edges           | Transitions between blocks.                                                |
| Number of Execution Paths | Sequence of blocks that are traversed during the execution of the program. |

Table 3.4: Software Complexity Metrics Description

#### 3.2.2.1 Cyclomatic Complexity

The first metric chosen to measure this was the cyclomatic complexity. Cyclomatic complexity measures the number of nested conditions within the code. The greater the number of conditions, the greater the complexity. By comparing the cyclomatic complexity before and after the fix, we are able to calculate the complexity change of the code, and estimate the effort needed to either increase or decrease the complexity of the code, the higher the complexity change, the higher the effort made by the developers.

While iterating through the modified files in the commit, we save the source code of the relevant files locally, making use of the `source_code` property of the file object, and calculate the cyclomatic complexity of the file using the `lizard` tool [25].

```

1 # Get modified file content from fix, save it in corresponding doc
  with open(file_save_local, "w") as outfile:
3     outfile.write(file.source_code)

5 # Calculate Execution Path metrics for Fix file
  fix_execution = execution_path_metrics(file_save_local)
7  fix_total_paths = next(fix_execution)
  fix_total_blocks = next(fix_execution)
9  fix_total_edges = next(fix_execution)
  fix_decision_forks = next(fix_execution)
11 fix_forks = next(fix_execution)

13 fix_info = cyclomatic_complexity(repo_name, file_save_local)

```

The `lizard` tool will retrieve the cyclomatic complexity of each individual function present in the source code. Afterwards, the sum of these complexities is made to determine the complexity of the overall file.

```

1 def file_cyclomatic_complexity(fix):
    fix_json = open(fix,)
3     fix_data = json.load(fix_json)

5     file_complexity = 0
    for function in fix_data:
7         file_complexity += fix_data[function]['cyclomatic complexity']
        fix_data['cyclomatic complexity'] = file_complexity
9     fix_data['net cyclomatic complexity'] = file_complexity

```

To calculate the difference in cyclomatic complexity, we also need to determine the cyclomatic complexity of the files before the security patch was committed. Using the `source_code_before` property of the file object, we can retrieve the source code of the file before the commit related to the fix.

If the file existed before the commit, the calculation is done following the same steps, as

mentioned above, by first saving the source code of the file locally and using the lizard tool to determine the cyclomatic complexity. If the file did not exist before the commit, we skip the calculation of the cyclomatic complexity.

```

1 # Get vulnerable files, save it in corresponding doc
  if file.source_code_before is not None:
3     with open(file_save_local, "w") as outfile:
        outfile.write(file.source_code_before)
5
        vulnerable_info = cyclomatic_complexity(repo_name, file_save_local)
7 else:
    vulnerable_info = None

```

The final step is to calculate the difference in cyclomatic complexity before and after the commit. In the case where the file existed before and after the commit, we take into account both cyclomatic complexities and calculate their difference. For the case where the file was created during the fix and did not exist beforehand, only the cyclomatic complexity of the new file will be considered.

```

  if fix_info is not None and vulnerable_info is not None:
2     compare_cyclomatix_complexity(fix_info, vulnerable_info)
  else:
4     # Calculate Cyclomatic Complexity when file exists only after commit
        file_cyclomatic_complexity(fix_info)

```

To compare the cyclomatic complexity between two files, we iterate through each individual function and compare the values before and after the fix. If the function was added during the fix, the complexity difference for that function will match the value of its complexity. Similarly, if a function has been deleted during the fix, the complexity difference will match the value of the complexity score of the function that was deleted.

```

1 # Functions have been added as part of the fix
  if num_fix_functions >= num_vuln_functions:
3     for function in fix_data:
        if function in vuln_data:
5         fix_data[function]['previous complexity'] = vuln_data[function]['
            cyclomatic complexity']

```

```

        fix_data[function]['complexity difference'] = fix_data[function]['
cyclomatic complexity'] - vuln_data[function]['cyclomatic complexity']
7     else:
        fix_data[function]['previous complexity'] = 0
9     fix_data[function]['complexity difference'] = fix_data[function]['
cyclomatic complexity']
# Functions have been deleted as part of the fix
11 else:
    for function in vuln_data:
13         if function in fix_data:
            fix_data[function]['previous cyclomatic complexity'] = vuln_data[
function]['cyclomatic complexity']
15         fix_data[function]['complexity difference'] = fix_data[function]['
cyclomatic complexity'] - vuln_data[function]['cyclomatic complexity']
            else:
17                 fix_data[function] = {}
                fix_data[function]['previous cyclomatic complexity'] = vuln_data[
function]['cyclomatic complexity']
19                 fix_data[function]['cyclomatic complexity'] = 0
                fix_data[function]['complexity difference'] = 0 - vuln_data[function
]['cyclomatic complexity']

```

### 3.2.2.2 Execution Path Metrics

Another set of metrics considered are related to the execution paths of each program modified during the fix. For this section, the following metrics were selected: the number of blocks, the number of edges, and the total number of execution paths.

A block is a sequence of instructions without branches except at the very end. By measuring the total number of blocks, we gain insight into the structural complexity of the program. The difference in the number of blocks after the security patch will indicate how much of this structural complexity was altered, the higher the difference, the more logic the developers had to reason about, the higher the effort required to code the security patch. It could also indicate whether new logic has been added or if previous logic was deleted by the security patch when used in conjunction with the remaining execution path metrics.

An edge represents a connection between logic units, in this case, a transition between blocks. The total number of edges of a program will represent the complexity of the flow of its execution. The higher the difference in the number of edges after the fix, the more the behavior of the program will change as new conditions might be introduced, meaning that developers spent more cognitive effort to implement this change.

Lastly, we have the total number of execution paths. A path is a sequence of blocks that are traversed during the execution of the program. By having the number of total execution routes and measuring the difference between before and after the fix, we can know if new paths were introduced by new logic or were removed from the program. Either case implies that there was necessary effort from the developers to make this change.

To measure this, we used the following code from the project `PyTA Project: Depth-First Search for Control Flow Graph` [26], first creating the control flow graph using the `astroid` library.

`Astroid` is a library for AST parsing, static analysis and inference whose main goal is to provide a common base representation of python source code [27].

Once the control flow graph is constructed, we can now perform a depth-first search to search for all possible execution paths in the program being analyzed.

```
def find_all_paths(cfg: ControlFlowGraph) -> list[list[int]]:
2     """
    Find all paths from the start block to the end block in the given control
    flow graph.
4     """
    return list(_dfs(cfg.start, [], set()))
```

To retrieve the number of blocks and edges, we use the `get_blocks()` and `get_edges()` functions to iterate through the `cfg` object previously, and for each `cfg` in the `cfgs` object, get the number of blocks and edges on each one.

```
1 # Create CFG for file after fix
   cfgs = create_cfg(file)
3 for cfg in cfgs.values():
    number_blocks = 0
5     edges = 0
```

```

7     for block in cfg.get_blocks():
            if block.id not in total_blocks:
9                 total_blocks.append(block.id)

11    for edge in cfg.get_edges()))
            for edge in cfg.get_edges():
13                edges += 1

```

### 3.3. Structural Churn Score

The Structural Churn Score is calculated through the mean of the absolute values of the metrics defined above.

$$\begin{aligned}
 \text{StructuralChurnScore} = & (\alpha|\text{LOCs}| + \beta|\text{Files}| + \gamma|\Delta\text{CC}| \\
 & + \delta|\Delta\text{Blocks}| + \epsilon|\Delta\text{Edges}| + \zeta|\Delta\text{Paths}|) / 6
 \end{aligned}
 \quad (3.1)$$

Where the LOCs and Files are the total number of lines and files modified during the fix, respectively and with  $\Delta$  being the difference between the value of each measured metric before and after the patch, i.e., between the vulnerable and fixed versions of the code.

$$\Delta\text{Metric} = \text{MetricValue}_{\text{fix}} - \text{MetricValue}_{\text{vulnerable}} \quad (3.2)$$

At this stage, all metrics share the same weight, meaning that no metric is more relevant than the other when it comes to determining the effort required to make the fix.

$$\alpha = \beta = \gamma = \delta = \epsilon = \zeta = 1 \quad (3.3)$$

```

1 def effort_calculation(total_lines, number_files, diff_complexity, diff_blocks,
    diff_edges, diff_paths, diff_forks):
    weight_lines = 1
3    weight_files = 1
    weight_complexity = 1
5    weight_blocks = 1
    weight_edges = 1
7    weight_paths = 1
    weight_forks = 1

```

9

```
score = (abs(total_lines)*weight_lines + number_files*weight_files + abs(  
diff_complexity)*weight_complexity + abs(diff_blocks)*weight_blocks + abs(  
diff_edges)*weight_edges + abs(diff_paths)*weight_paths) / 6
```

11

```
return score
```

### 3.4. Running the Script

To standardize the execution process, we created **input files** for each run of the analysis script. Each file specifies the target repository name and the corresponding fix commit identifier. An example input file is shown below:

---

```
aiohttp  
e8d774f635dc6d1cd3174d0e38891da5de0e2b6a
```

---

Before running the script, the target repository must be cloned locally. This step allows repeated analysis to be performed in an offline environment, ensuring reproducibility and faster access to source files. Once the repository is available locally, the script can be executed from the command line followed by invoking the Python command with the script name and piping the input file as shown below:

---

```
python script.py < input.txt
```

---

## 4. Results

In the following pages, we present the answers to the proposed research questions.

### 4.1. What kinds of structural changes are typical in Python security patches?

To establish a quantitative baseline of structural changes introduced by Python security patches, we analyzed 100 patches and computed delta values for 6 dimensions: lines of code ( $\Delta\text{LOCS}$ ), number of modified files ( $\Delta\text{Files}$ ), cyclomatic complexity ( $\Delta\text{CC}$ ), and control-flow-level changes ( $\Delta\text{Blocks}$ ,  $\Delta\text{Edges}$ ,  $\Delta\text{Paths}$ ).

**Quantitative Baseline for Patch Structure.** We computed descriptive statistics for all six metrics (Table 4.1), including a composite structural change score defined as the sum of the absolute deltas.

| Metric  | $\Delta\text{LOC}$ | $\Delta\text{Files}$ | $\Delta\text{Cyclomatic}$ | $\Delta\text{Blocks}$ | $\Delta\text{Edges}$ | $\Delta\text{Paths}$ |
|---------|--------------------|----------------------|---------------------------|-----------------------|----------------------|----------------------|
| Mean    | 72.28              | 1.83                 | 3.83                      | 1.95                  | 7.85                 | 107.25               |
| Std Dev | 255.25             | 1.54                 | 13.60                     | 10.61                 | 26.94                | 776.24               |
| Min     | 1.00               | 1.00                 | -7.00                     | -6.00                 | -19.00               | -620.00              |
| 25%     | 8.75               | 1.00                 | 0.00                      | 0.00                  | 0.00                 | 0.00                 |
| Median  | 21.00              | 1.00                 | 1.00                      | 0.00                  | 3.00                 | 1.00                 |
| 75%     | 58.25              | 2.00                 | 4.00                      | 1.00                  | 7.00                 | 3.50                 |
| Max     | 2451.00            | 10.00                | 129.00                    | 104.00                | 246.00               | 5848.00              |

Table 4.1: Descriptive statistics of structural changes in Python security patches

The median Python security patch involves 21 lines of codes changed, 1 modified file, a 1-point increase in in Cyclomatic Complexity, 3 new edges in the control-glow graph and 1 added execution path, with no change in the number of basic blocks. These values constitute a quantitative structural baseline for Python security patches.

While median values suggest that most patches are relatively small and localized, the high standard deviations and maximum values in Table 4.1 indicate substantial variability across the dataset. For example, although the median change in Cyclomatic Complexity is just +1, one patch showed an increase of +129. Similarly, changes in execution paths ranged from 620 to +5848. These outliers illustrate that a subset of patches can involve extensive structural changes, significantly exceeding the typical baseline.

Patches that significantly exceed median values can be flagged as above-baseline, potentially indicating increased review or testing effort. While we do not measure developer

effort directly, this structural profile serves as a **proxy for control-flow growth and patch complexity**, which are known contributors to maintenance and testing complexity. These insights can inform tooling, prioritization, and future work on effort-aware patch triage in software security workflows.

#### RQ1: What kind of structural changes are typical in Python security patches?

Python security patches typically introduce **moderate structural changes**.

- Most patches (89 security patches) modify **1–3 files**.
- The median number of lines modified in a security patch is around **20 lines of code**.
- Structural growth is seen in metrics like **Blocks**, **Edges**, and **Paths**, but often without adding entirely new code regions.
- Notably, **66%** of patches leave the number of blocks unchanged—suggesting that patches often **modify existing logic**, rather than add new blocks.

**Conclusion:** Security patches frequently alter logic internally but do not always involve large-scale code additions.

## 4.2. Do security patches tend to increase or decrease control-flow complexity?

To assess whether Python security patches typically tend to harden program logic by introducing new flow structures, or simplify it by removing paths, we analyzed directional changes in three control-flow metrics across 100 patches: number of blocks ( $\Delta\text{Blocks}$ ), control-flow edges ( $\Delta\text{Edges}$ ), and execution paths ( $\Delta\text{Paths}$ ).

| Change Type      | $\Delta\text{Blocks}$ | $\Delta\text{Edges}$ | $\Delta\text{Paths}$ | $\Delta\text{CC}$ |
|------------------|-----------------------|----------------------|----------------------|-------------------|
| <b>Increased</b> | 29                    | 58                   | 57                   | 62                |
| <b>Decreased</b> | 5                     | 8                    | 9                    | 5                 |
| <b>Unchanged</b> | 66                    | 34                   | 33                   | 33                |

Table 4.2: Directional changes in control-flow metrics across 100 Python security patches

**Structural Trends.** For each patch, we recorded whether these metrics increased, decreased, or remained unchanged. Table 4.2 summarizes the proportion of patches where each metric increased, decreased, or remained unchanged. As expected, all patches involved changes to at least one file and a nonzero number of lines of code.

The results show that most Python security patches (62%) introduce structural growth and complexity, with an average complexity increase of 6.44, typically by adding new control-flow edges (58%), blocks (67%) and execution paths (57%). These results show that security patches are more likely to **introduce** new control-flow complexity than to remove it. The most frequent structural change is the addition of control-flow edges (58%), followed closely by new execution paths (57%). Changes to the number of basic blocks are less common—remaining unchanged in 66% of patches—which suggests that many patches **modify existing blocks** (e.g., by adding conditions) rather than introducing entirely new code regions.

Only a small subset of patches (5–9%) showed any **reduction** in control-flow metrics, indicating that simplification is rare. These patterns support the hypothesis that **Python security patches typically harden program logic** by adding control structures (e.g., guards, input checks) rather than removing logic.

Since additional paths and edges imply more cases to reason about, these structural changes may also correspond to higher testing and review burden, especially when verifying patch correctness under multiple conditions.

#### RQ2: Do security patches tend to increase or decrease control-flow complexity?

Security patches tend to **increase control-flow complexity**.

- Cyclomatic Complexity increased in **62%** of patches, with an average complexity increase of **6.44**.
- Edges and Paths increased in **58%** and **57%** respectively.
- Reductions in these metrics were rare (only **5–9%** of patches).

**Conclusion:** Python security patches often **harden the logic** by adding conditions or checks, supporting the assumption that patches tend to make code more complex.

### 4.3. Are larger structural changes ( $\Delta$ Blocks, $\Delta$ Edges, $\Delta$ Files) associated with increased control-flow complexity ( $\Delta$ CC, $\Delta$ Paths)?

To evaluate whether the size of a security patch is associated with its control-flow complexity, we analyzed the correlations between size-related metrics ( $\Delta$ LOCs,  $\Delta$ Files) and complexity-related metrics ( $\Delta$ Cyclomatic Complexity,  $\Delta$ Blocks,  $\Delta$ Edges,  $\Delta$ Paths).

**Understanding Relationships Between Metrics.** Figure 4.1 shows a scatter matrix that helps us understand how different metrics relate to one another. It includes six metrics measured across 100 Python security patches.

On the diagonal, we see individual distributions of each metric. Most of these distributions are skewed to the right, meaning that **most patches are small or simple**, but a few are very large or complex (outliers). For example, some patches have over 2,000 changed lines or over 6,000 possible paths through the program. The number of Files shows a different pattern, with values grouped around 1, 23, and 4+, which reflects the way patches are grouped later in the analysis.

The scatter plots off the diagonal show how pairs of metrics vary together:

- **Strong relationships among complexity metrics:** Cyclomatic Complexity, Blocks, Edges, and Paths tend to grow together. When a patch adds more blocks, it usually also adds more edges and more execution paths, resulting in higher complexity. These relationships appear as tight, upward trends in the scatter plots.
- **LOC is related, but less strongly:** Patches that touch more lines of code often introduce more complexity, but the relationship is not perfect. Some small patches are quite complex, and some large patches are relatively simple. This means that LOC alone is not a reliable way to estimate complexity.
- **Files are weakly related to everything else:** The number of files touched by a patch shows very little correlation with other metrics. Many patches that touch the same number of files have very different LOC or complexity values. This suggests that just counting files is not enough to understand how difficult or complex a patch really is.

In summary, the scatter matrix shows that internal control-flow complexity (measured by CC, Blocks, Edges, and Paths) is tightly connected. On the other hand, size-related metrics like LOC or number of Files give us a less complete picture of a patch's structural depth or difficulty.

**Heatmap of Metric Correlations.** To complement the visual patterns seen in the scatter matrix, Figure 4.2 presents a correlation heatmap showing the Pearson correlation

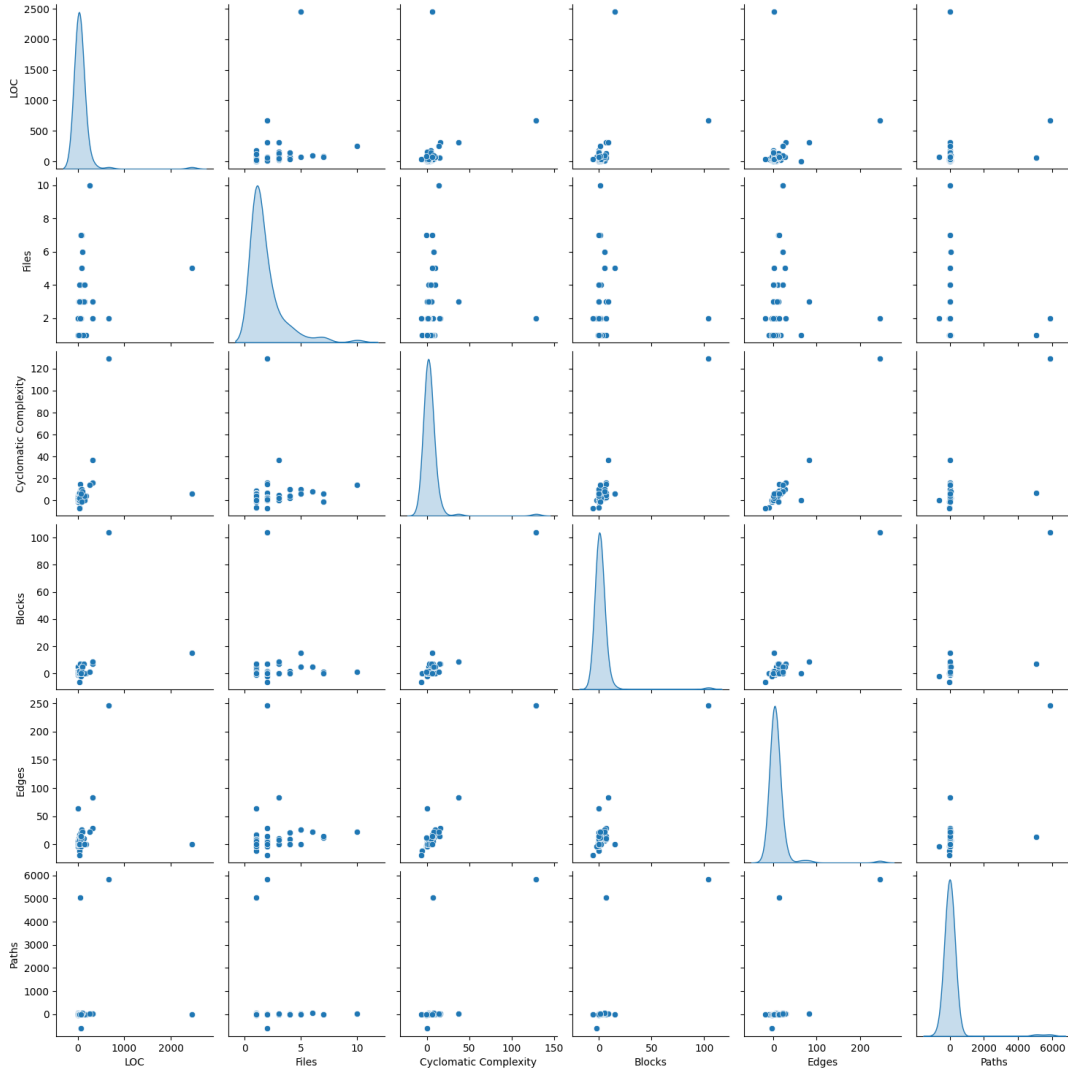


Figure 4.1: Scatter Matrix for measured Metrics

coefficients between all pairs of measured metrics. This allows us to confirm and quantify the relationships discussed previously.

The most prominent feature of the heatmap is a tightly clustered block of high correlations among the four control-flow complexity metrics:

- **Cyclomatic Complexity (CC)** correlates almost perfectly with both **Blocks** and **Edges** ( $r = 0.96$ ), and strongly with **Paths** ( $r = 0.73$ ).
- **Blocks** and **Edges** are also highly correlated ( $r = 0.92$ ), and each shows a strong positive correlation with **Paths** ( $r = 0.77$  and  $r = 0.72$ , respectively).

This confirms that these four metrics capture overlapping aspects of a patch’s control-flow complexity: when one increases, the others typically do as well. Their high mutual

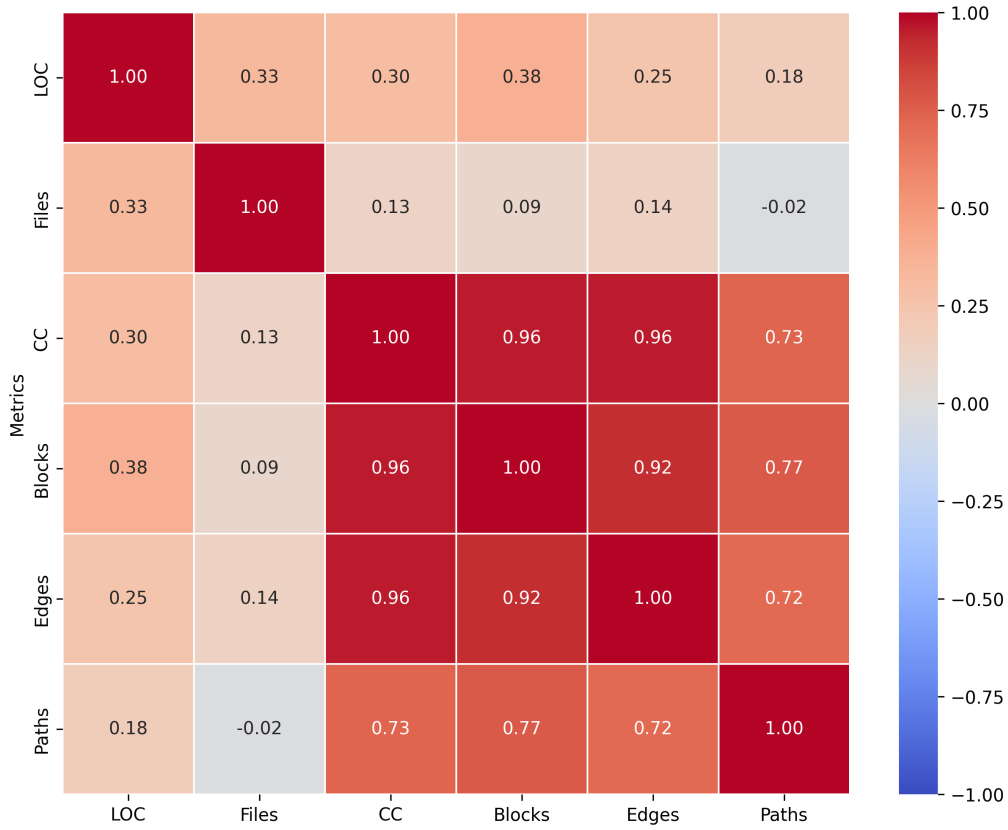


Figure 4.2: Correlation Heatmap between Software Metrics

correlations also suggest that they might be partially redundant, though each contributes a slightly different view of structural complexity.

In contrast, size-related metrics show much weaker associations:

- **LOC** has moderate correlations with the complexity metrics, ranging from  $r = 0.18$  (with Paths) to  $r = 0.38$  (with Blocks). This confirms that while larger patches tend to be more complex, the relationship is not strong enough to treat LOC as a standalone complexity indicator.
- **Files** shows minimal correlation with any other metric, including a near-zero correlation with Paths ( $r = -0.02$ ) and only weak positive correlation with LOC ( $r = 0.33$ ). This reinforces the idea that the number of files affected in a patch does not meaningfully predict its internal complexity or patching effort.

Overall, the heatmap reinforces the interpretation drawn from the scatter matrix: **internal control-flow metrics are closely tied and form the core of structural complexity**,

while superficial indicators like file count or lines of code provide only limited insight into the true effort required to implement a patch.

**Boxplots: Files vs. Complexity Metrics.** To investigate whether the breadth of a patch (i.e., the number of files affected) is indicative of its structural complexity, we grouped the 100 security patches into three categories: 1) **Single-file patches** (1 file modified); 2) **Small multi-file patches** (2–3 files modified); 3) **Large multi-file patches** (4 or more files modified).

We then plotted the distribution of each control-flow metric within these groups: Cyclomatic Complexity, Blocks, Edges, and Paths. The resulting boxplots are shown in Figures 4.3, 4.4, 4.5 and 4.6.

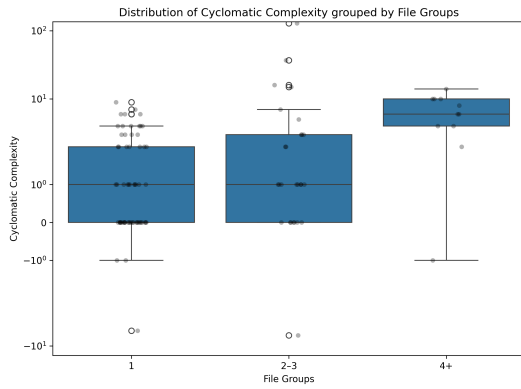


Figure 4.3: Cyclomatic Complexity by File Count

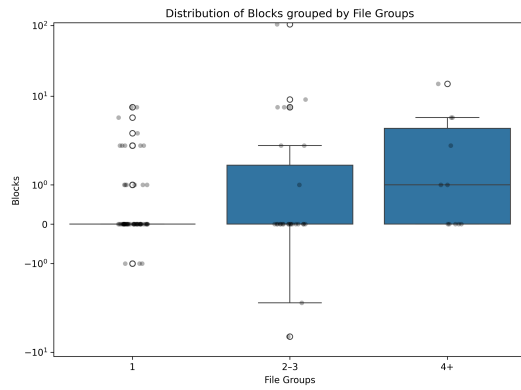


Figure 4.4: Blocks by File Count

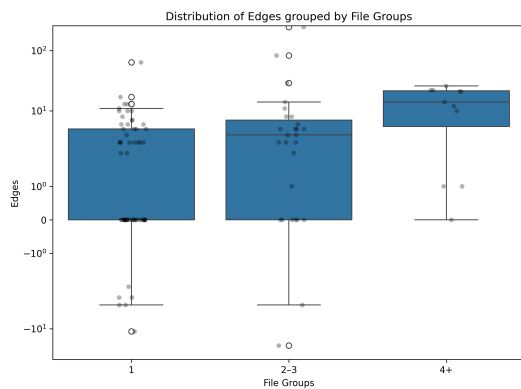


Figure 4.5: Edges by File Count

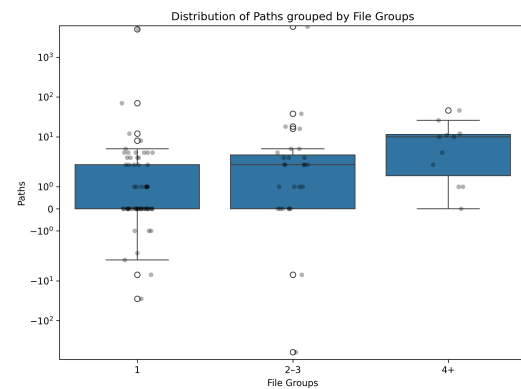


Figure 4.6: Paths by File Count

The results of the boxplots reveal several important patterns. First, in terms of **Cyclomatic Complexity**, the median values remain relatively stable between the single-file and 2–3 file patch groups. However, patches that span four or more files tend to exhibit a noticeable increase in complexity. This is evidenced by a shift in the median

value and a broader interquartile range, reflecting more variability in how complex these patches become. Nonetheless, even in the smaller patch groups, some outliers exhibit very high complexity. This suggests that even small patches can introduce substantial logical changes, possibly due to intricate control flow logic rather than breadth alone. Looking at the **number of basic blocks**, we observe that single-file patches often leave the number of blocks unchanged, with a median of zero. However, several outliers in this group show significant changes, implying that even within limited scope, developers may restructure core parts of the logic. As we move to patches that touch more files, the median and overall spread of block changes increase, confirming that **broader patches tend to be more structurally invasive**. Finally, the **number of control-flow edges and execution paths** follows a pattern similar to complexity and blocks. Patches affecting only one to three files typically show modest changes, while those modifying four or more files display higher median values and greater variability. This further supports the idea that larger patches are more likely to increase structural complexity, but also highlights that smaller patches are not necessarily simple, a subset of them still alters execution flow in nontrivial ways. In summary, the number of files affected in a patch is loosely associated with complexity trends, but it is not a reliable standalone predictor. Patching effort and structural impact must be assessed using internal control-flow metrics directly, as even narrow patches can introduce deep and complex changes.

**Correlation with Structural Churn Score.** To understand which factors most influence the computed structural effort of a security patch, we calculated the Pearson correlation between each individual metric and the overall Structural Churn Score, a composite value computed as the mean of structural and size-related changes introduced by each patch. Figure 4.7 summarizes these results.

The results reveal a clear distinction between size-related metrics (Lines of Code, Files) and control-flow complexity metrics (Cyclomatic Complexity, Blocks, Edges, Paths). While the number of changed lines (LOC) shows a moderate correlation with effort ( $r = 0.46$ ), the number of affected files is nearly uncorrelated ( $r = 0.079$ ), reinforcing our earlier observation that patch breadth is a poor indicator of patching effort.

In contrast, the internal structural complexity metrics show much stronger associations with the computed effort score. The number of modified execution paths exhibits the strongest correlation ( $r = 0.94$ ), followed by the number of blocks ( $r = 0.82$ ), Cyclomatic Complexity ( $r = 0.76$ ), and control-flow edges ( $r = 0.72$ ). These strong correlations

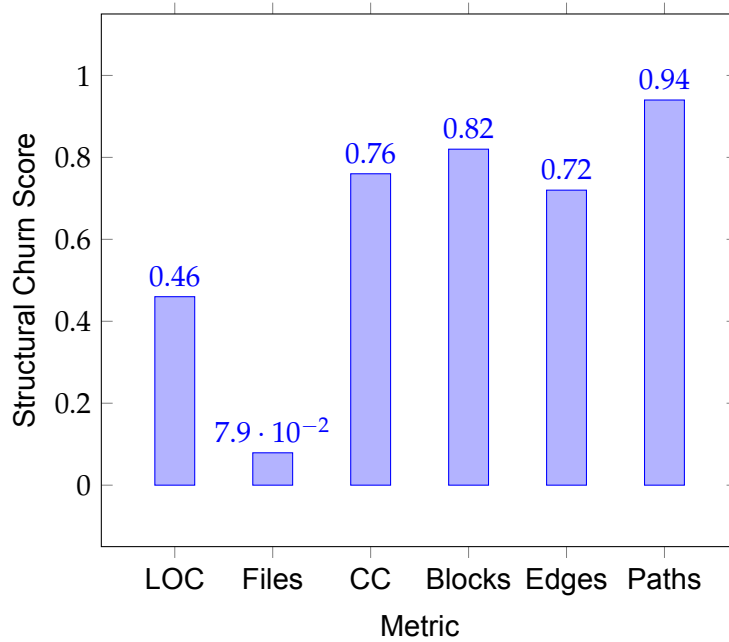


Figure 4.7: Correlation between Structural Churn Score and Software Metrics

suggest that changes which alter control flow logic are the primary contributors to patch effort.

Importantly, this also shows that patch size alone cannot explain effort. Even patches that touch a small number of lines or files may involve substantial logical restructuring, reflected in higher path or complexity deltas, and thus incur high effort scores. This observation supports our broader claim that structural control-flow complexity is a more accurate indicator of patching effort than size-based metrics alone.

**RQ3: Are larger structural changes associated with increased control-flow complexity?**

Yes, particularly for internal control-flow metrics.

- High correlations between **Blocks**, **Edges**, and **Cyclomatic Complexity** ( $r \approx 0.92\text{--}0.96$ ).
- **Paths** also correlates moderately well ( $r = 0.72\text{--}0.77$ ).
- **LOC** shows only moderate correlation with complexity ( $r \approx 0.25\text{--}0.38$ ).
- **Number of files** shows weak or no correlation (e.g.,  $r = -0.02$  with Paths).

**Conclusion:** Structural complexity is better explained by internal changes to logic rather than size or file count.

## 5. Discussion and Implications

This chapter reflects on the key findings of our study and their broader implications.

### 5.1. Effort Beyond Size: Revisiting the Metrics

Our study shows that patching effort is not well explained by traditional size-based metrics such as Lines of Code (LOC) or number of files changed. Although LOC has a moderate correlation with effort ( $r = 0.46$ ), the number of modified files has minimal predictive power ( $r = 0.079$ ). In contrast, internal control-flow complexity metrics, particularly the number of execution paths ( $r = 0.94$ ), blocks ( $r = 0.82$ ), and edges ( $r = 0.72$ ), demonstrate strong alignment with the computed Structural Churn Score. This empirical evidence supports a structural view of patch difficulty: **patches that introduce more branches, paths, and logical decisions are significantly harder to reason about, test, and validate**, regardless of their size. Thus, any system that aims to prioritize vulnerabilities realistically should factor in this form of structural effort.

### 5.2. Prioritization with Effort-Aware Triage

Most vulnerability management frameworks prioritize alerts based solely on severity. However, this approach overlooks developer-side concerns such as implementation complexity and review burden. Our findings suggest that two vulnerabilities with identical severity can have vastly different patching effort.

The proposed **Structural Churn Score** introduces a second dimension to alert prioritization: the developer-side cost of applying a fix. Incorporating this score into triage frameworks allows teams to:

- Deprioritize alerts that are low-effort and quick to patch.
- Flag alerts with high structural effort for further review or pairing with more experienced developers.
- Allocate patching resources more effectively, balancing severity with fix feasibility.

This effort-aware perspective enhances vulnerability management pipelines, particularly in organizations with limited developer bandwidth.

### 5.3. Quality Gates for Automated Patches

As the field of automated vulnerability repair grows, there is a need to ensure that machine-generated patches remain human-manageable. Our Structural Churn Score provides a mechanism for post-generation sanity checks: patches with anomalously high structural complexity can be flagged as candidates for revision or manual validation. This offers an objective signal to detect overly complex, nonsensical, or high-risk fixes even when generated by high-performing LLMs or automated repair tools.

### 5.4. Language-Agnostic Expansion

Although our study focused on Python, the core metrics used in the Structural Churn Score, control-flow blocks, edges, paths, and cyclomatic complexity, are language-agnostic. This creates opportunities to generalize the approach to other ecosystems (e.g., Java, C++, Rust), contingent on access to structural code analyzers for those languages.

Tools like `pyDriller`, used in this thesis, can be substituted with equivalent parsing frameworks (e.g., `JavaParser`, Clang AST tools), allowing this metric to scale across modern security patch corpora.

Another avenue for investigation concerns whether the structural baseline of a security patch is inherently language-dependent. Programming languages are typically employed in distinct operational contexts and exhibit different vulnerability profiles, with certain classes of security weaknesses being more prevalent in some languages than in others. For instance, languages frequently used in frontend development, such as JavaScript and C#, tend to encounter vulnerability types that differ from those commonly associated with backend-oriented languages like Java and C. As different categories of vulnerabilities demand different forms of remediation, it's expected that the structural characteristics of the resulting patches may also vary. This raises the question of whether the structural baseline of a security patch is influenced by the programming language in which it is implemented, and to what extent such variability should be accounted for in vulnerability management research.

### 5.5. Fine-Tuning Metric Weighting

Currently, the Structural Churn Score treats each structural metric equally through a simple average. While this yields promising results, our correlation analysis shows that some metrics (e.g., Paths, Blocks) are more informative than others. This opens the door for future versions of the Structural Churn Score to be fine-tuned, either through regression

modeling or optimization-based weighting, to better align with developer-perceived effort or patch failure rates. Such improvements would increase the precision of the metric while preserving its interpretability.

## 5.6. Practical Caveats and Limitations

There are some shortcomings with the way the Structural Churn Score is calculated that are worth pointing out.

Due to time constraints, the test cases employed in this study were manually selected and reviewed. This process was inherently time-consuming, which limited the number of test cases included in the analysis. Future work should prioritize the automation of this selection procedure, enabling the inclusion of a substantially larger dataset and enhancing the overall robustness and accuracy of the results.

Another practical caveat is related to the comments left in the code by developers. As it stands, the number of modified lines of code with the security patch is calculated by having not only the lines of code itself but also comments left for future developers to read.

An argument can be made that commenting code is a good practice that should be followed, and thus the effort required to write out those comments should be taken into account. The importance of comments was also shown by Fry *et al.* [22], strengthening this line of thought.

The reverse argument can also be made that comments should not account for the total number of modified lines of code. In this case, the number of modified lines of code in a security patch would decrease, making the weight of the complexity related metrics even more relevant in the calculation of the Structural Churn Score. To achieve this, a new function has to be added to the current program used to calculate the Structural Churn Score to filter out any comment lines from the final number of modified lines of code.

Some examples of test cases used where there is a substantial amount of comments in the code are present in Table 5.1.

Table 5.1: Test cases where comments make up a big part of the LOC

| ID                  | LOC |
|---------------------|-----|
| GHSA-4qjh-9fv9-r85r | 258 |
| GHSA-5qwj-342r-h886 | 317 |

Another point to consider is that the current program does not analyze files that contain test files devised to test new functionalities added with each security patch. Should the effort to code these tests be considered? If so, the current filter should be removed from the program.

## 5.7. Summary

In summary, our study contributes both empirical evidence and actionable insights into the nature of patching effort. We demonstrate that security patches often increase control-flow complexity, and that this complexity, not size, best explains developer-side difficulty.

By operationalizing this insight into a metric (Structural Churn Score), we propose a concrete step toward more informed, effort-aware alert prioritization. Our findings encourage a shift from purely severity-driven triage toward a dual-axis model that considers both impact and cost, paving the way for smarter workflows in secure software maintenance.

## 6. Conclusion

With the adoption of a closely integrated security posture with software development, the number of software vulnerabilities detected will continue to increase, increasing the burden and workload on developers and security auditors. The prioritization of remediation of said vulnerabilities will be paramount to enact an effective security program.

We've analyzed 100 Python security patches from open-source software projects, extracting metrics such as the number of modified lines ( $\Delta\text{LOCs}$ ), number of files ( $\Delta\text{Files}$ ), cyclomatic complexity ( $\Delta\text{CC}$ ), and changes in control-flow structure ( $\Delta\text{Blocks}$ ,  $\Delta\text{Edges}$ ,  $\Delta\text{Paths}$ ) and combined these into a **Structural Churn Score** that serves as a quantitative proxy for patch complexity.

A quantitative structural baseline for Python security patches was found. The median Python security patch involves 21 lines of codes changed, 1 modified file, a 1-point increase in in Cyclomatic Complexity, 3 new edges in the control-flow graph and 1 added execution path, with no change in the number of basic blocks. We've concluded that security patches frequently alter logic internally but do not always involve large-scale code additions.

The results also show that most Python security patches (62%) introduce structural growth and complexity, typically by adding new control-flow edges (58%), blocks (67%) and execution paths (57%), showing that Python security patches often harden the logic by adding conditions or checks, supporting the assumption that patches tend to make code more complex.

It was also found that larger structural changes associated with increased control-flow complexity, particularly for internal control-flow metrics, with high correlations found between Blocks, Edges, and Cyclomatic Complexity ( $r \approx 0.92\text{--}0.96$ ), with Paths also correlating moderately well ( $r = 0.72\text{--}0.77$ ), while LOC showing only moderate correlation with complexity ( $r \approx 0.25\text{--}0.38$ ), and the number of files showing weak or no correlation ( $r = -0.02$  with Paths). Give this, we can conclude that structural complexity is better explained by internal changes to logic rather than size or file count.

We also observed that **internal control-flow metrics are closely tied and form the core of structural complexity**, while superficial indicators like file count or lines of code provide only limited insight into the true effort required to implement a patch, a finding

later confirmed again by the correlation between the complexity metrics, especially the number of blocks and paths, and the effort required being stronger than the correlation of the effort with size metrics, meaning that the complexity of the security patch is more telling of the overall effort needed than the size of the security patch.

A concrete step toward more informed, effort-aware alert prioritization was proposed with the adoption of a new metric (Structural Churn Score), encouraging a shift from purely severity-driven triage toward a dual-axis model that considers both impact and cost.

## References

- [1] T. W. Thomas, M. Tabassum, B. Chu, and H. Lipford, "Security during application development," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, Apr. 2018, p. 1–12. [Online]. Available: <https://doi.org/10.1145/3173574.3173836> [Cited on page 1.]
- [2] W. N. Behutiye, P. Rodríguez, M. Oivo, and A. Tosun, "Analyzing the concept of technical debt in the context of agile software development: A systematic literature review," *Information and Software Technology*, vol. 82, p. 139–158, Feb. 2017. [Online]. Available: <https://doi.org/10.1016/j.infsof.2016.10.004> [Cited on page 1.]
- [3] M. Siavvas, D. Tsoukalas, M. Jankovic, D. Kehagias, and D. Tzovaras, "Technical debt as an indicator of software security risk: a machine learning approach for software development enterprises," *Enterprise Information Systems*, vol. 16, no. 5, Sep. 2020. [Online]. Available: <https://doi.org/10.1080/17517575.2020.1824017> [Cited on page 1.]
- [4] E. Iannone, R. Guadagni, F. Ferrucci, A. De Lucia, and F. Palomba, "The secret life of software vulnerabilities: A large-scale empirical study," *IEEE Transactions on Software Engineering*, vol. 49, no. 1, p. 44–63, Jan. 2023. [Online]. Available: <https://doi.org/10.1109/tse.2022.3140868> [Cited on page 1.]
- [5] [Online]. Available: <https://safecomputing.umich.edu/protect-the-u/protect-your-unit/vulnerability-management/remediation> [Cited on page 1.]
- [6] [Online]. Available: <https://security.utoronto.ca/governance/guidelines/vulnerability-management/> [Cited on page 1.]
- [7] Y. Jiang, N. Oo, Q. Meng, H. W. Lim, and B. Sikdar, "A survey on vulnerability prioritization: Taxonomy, metrics, and research challenges," *CoRR*, vol. abs/2502.11070, 2025, arXiv:2502.11070. [Online]. Available: <https://arxiv.org/abs/2502.11070> [Cited on page 1.]
- [8] L. Othmane, G. Chehrazi, E. Bodden, P. Tsalovski, A. D. Brucker, and P. Miseldine, "Factors impacting the effort required to fix security vulnerabilities," in *Proceedings of the 18th International Conference on Information Security - Volume 9290*, ser. ISC

2015. Berlin, Heidelberg: Springer-Verlag, 2015, p. 102–119. [Online]. Available: [https://doi.org/10.1007/978-3-319-23318-5\\_6](https://doi.org/10.1007/978-3-319-23318-5_6) [Cited on pages 2 and 7.]
- [9] P. R. de Bassi, G. M. P. Wanderley, P. H. Banali, and E. C. Paraiso, “Measuring developers’ contribution in source code using quality metrics,” in *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design ((CSCWD))*. IEEE, May 2018, p. 39–44. [Online]. Available: <https://doi.org/10.1109/cscwd.2018.8465320> [Cited on pages 4 and 6.]
- [10] H. Myrbakken and R. Colomo-Palacios, “Devsecops: A multivocal literature review,” in *Software Process Improvement and Capability Determination*, 09 2017, pp. 17–29. [Cited on page 4.]
- [11] R. Wijayarathne, “Security in computing (third edition) by charles pfleeger and shari pfleeger (2004), pearson education (singapore) pte. ltd., india branch, isbn 81-297-0042-5. 746 pages. index. bibliography,” *Journal of the National Science Foundation of Sri Lanka*, vol. 34, no. 2, p. 109, Jun. 2006. [Online]. Available: <https://doi.org/10.4038/jnsfsr.v34i2.2091> [Cited on page 4.]
- [12] T. Muske and A. Serebrenik, “Survey of approaches for postprocessing of static analysis alarms,” *ACM Computing Surveys*, vol. 55, no. 3, p. 1–39, Feb. 2022. [Online]. Available: <https://doi.org/10.1145/3494521> [Cited on page 4.]
- [13] —, “Survey of approaches for handling static analysis alarms,” in *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, Oct. 2016, p. 157–166. [Online]. Available: <https://doi.org/10.1109/scam.2016.25> [Cited on page 4.]
- [14] V. Kumar, M. Anjum, V. Agarwal, and P. K. Kapur, *A Hybrid Approach for Evaluation and Prioritization of Software Vulnerabilities*. Cham: Springer International Publishing, 2023, pp. 39–51. [Online]. Available: [https://doi.org/10.1007/978-3-031-05347-4\\_3](https://doi.org/10.1007/978-3-031-05347-4_3) [Cited on page 5.]
- [15] R. Sharma, R. Sibal, and S. Sabharwal, “Software vulnerability prioritization using vulnerability description,” *International Journal of System Assurance Engineering and Management*, vol. 12, no. 1, p. 58–64, Jul. 2020. [Online]. Available: <https://doi.org/10.1007/s13198-020-01021-7> [Cited on page 5.]
- [16] Z. Han, X. Li, Z. Xing, H. Liu, and Z. Feng, “Learning to predict severity of software vulnerability using only vulnerability description,” in *2017 IEEE International*

- Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Sep. 2017, p. 125–136. [Online]. Available: <https://doi.org/10.1109/icsme.2017.52> [Cited on page 5.]
- [17] K. A. Farris, A. Shah, G. Cybenko, R. Ganesan, and S. Jajodia, “Vulcon: A system for vulnerability prioritization, mitigation, and management,” *ACM Trans. Priv. Secur.*, vol. 21, no. 4, Jun. 2018. [Online]. Available: <https://doi.org/10.1145/3196884> [Cited on page 6.]
- [18] [Online]. Available: <https://nvd.nist.gov/vuln-metrics/cvss> [Cited on page 6.]
- [19] T. Honglei, S. Wei, and Z. Yanan, “The research on software metrics and software complexity metrics,” in *2009 International Forum on Computer Science-Technology and Applications*, vol. 1, 2009, pp. 131–136. [Cited on page 6.]
- [20] J. Munson and S. Elbaum, “Code churn: a measure for estimating the impact of code change,” in *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. IEEE Comput. Soc, p. 24–31. [Online]. Available: <https://doi.org/10.1109/icsm.1998.738486> [Cited on page 7.]
- [21] S. Misra, M. Koyuncu, M. Crasso, C. Mateos, and A. Zunino, “A suite of cognitive complexity metrics,” in *Computational Science and Its Applications – ICCSA 2012*, B. Murgante, O. Gervasi, S. Misra, N. Nedjah, A. M. A. C. Rocha, D. Taniar, and B. O. Apduhan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 234–247. [Cited on page 7.]
- [22] Z. P. Fry, B. Landau, and W. Weimer, “A human study of patch maintainability,” in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*. ACM, Jul. 2012, p. 177–187. [Online]. Available: <https://doi.org/10.1145/2338965.2336775> [Cited on pages 8 and 32.]
- [23] [Online]. Available: <https://osv.dev/> [Cited on page 9.]
- [24] D. Spadini, M. Aniche, and A. Bacchelli, *PyDriller: Python Framework for Mining Software Repositories*, 2018. [Cited on page 12.]
- [25] [Online]. Available: <https://github.com/terryyin/lizard> [Cited on page 15.]
- [26] [Online]. Available: <https://raineyang.hashnode.dev/pyta-project-depth-first-search-for-control-flow-graph> [Cited on page 18.]

[27] [Online]. Available: <https://pylint.readthedocs.io/projects/astroid/en/latest/> [Cited on page 18.]

# Appendix

## .1. Dataset

| ID                   | Severity | CWE      | Fix                                                                                                                                                                                                                               |
|----------------------|----------|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GHSA-9548-qrrj-x5pj  | 1,7      | CWE-444  | <a href="https://github.com/aio-libs/aiohttp/commit/e8d774f635dc6d1cd3174d0e38891da5de0e2b6a">https://github.com/aio-libs/aiohttp/commit/e8d774f635dc6d1cd3174d0e38891da5de0e2b6a</a>                                             |
| GHSA-g92j-qhmq-64v2  | 1,8      | CWE-200  | <a href="https://github.com/getsentry/sentry-python/commit/763e40aa4cb57eccd467f48f78f335c87e9bdf">https://github.com/getsentry/sentry-python/commit/763e40aa4cb57eccd467f48f78f335c87e9bdf</a>                                   |
| GHSA-v7ym-rhmq-8j2r  | 1,9      | CWE-602  | <a href="https://github.com/ethyca/fides/commit/ce664da46ab7f86d29583ebc34f2ff776f0aa6c2">https://github.com/ethyca/fides/commit/ce664da46ab7f86d29583ebc34f2ff776f0aa6c2</a>                                                     |
| GHSA-4qgf-9m5c-w2c5  | 2,1      | CWE-359  | <a href="https://github.com/WeblateOrg/weblate/commit/020b2905e4d001c1ff24525740ed16cd63621b5f62">https://github.com/WeblateOrg/weblate/commit/020b2905e4d001c1ff24525740ed16cd63621b5f62</a>                                     |
| GHSA-j79p-67mm-5g87  | 2,1      | CWE-862  | <a href="https://github.com/openedx/xblock-lti-consumer/commit/252f94bd182cd0962af9251015930cb55ec515d7">https://github.com/openedx/xblock-lti-consumer/commit/252f94bd182cd0962af9251015930cb55ec515d7</a>                       |
| GHSA-fm6c-f59h-7mmg  | 2,3      | CWE-502  | <a href="https://github.com/modelscope/ms-swift/commit/b3418ed9b050dc079553c275c5ed14cfb2b66cf7">https://github.com/modelscope/ms-swift/commit/b3418ed9b050dc079553c275c5ed14cfb2b66cf7</a>                                       |
| GHSA-4qjh-9fv9-r85r  | 2,6      | CWE-208  | <a href="https://github.com/vllm-project/vllm/commit/77073c77bc2006eb80ea6d5128f076f56ec6f54f">https://github.com/vllm-project/vllm/commit/77073c77bc2006eb80ea6d5128f076f56ec6f54f</a>                                           |
| GHSA-hhfj-fwrv-87w7  | 2,7      | CWE-20   | <a href="https://github.com/sigstore/sigstore-python/commit/300b502ae99ebfaace124f1f4e422a6a669389cf">https://github.com/sigstore/sigstore-python/commit/300b502ae99ebfaace124f1f4e422a6a669389cf</a>                             |
| GHSA-7mpr-5m44-h73r  | 2,9      | CWE-1284 | <a href="https://github.com/matthewwithanm/python-markdownify/commit/959561879693b04a576f99c6733b50b01186aa08">https://github.com/matthewwithanm/python-markdownify/commit/959561879693b04a576f99c6733b50b01186aa08</a>           |
| GHSA-674p-xv2x-rf3g  | 3,7      | CWE-117  | <a href="https://github.com/litestar-org/litestar/commit/03b5813d4f448dd710af9ba6252d798cb9cf087f">https://github.com/litestar-org/litestar/commit/03b5813d4f448dd710af9ba6252d798cb9cf087f</a>                                   |
| GHSA-5xh2-23cc-5jc6  | 3,7      | CWE-351  | <a href="https://github.com/strawberry-graphql/strawberry/commit/526eb82b70451c0e59d5a71ae9b739f59974bd8">https://github.com/strawberry-graphql/strawberry/commit/526eb82b70451c0e59d5a71ae9b739f59974bd8</a>                     |
| GHSA-c65p-x677-fgje  | 4,2      | CWE-1288 | <a href="https://github.com/vllm-project/vllm/commit/99404f53c72965b41558aceb1bc238087f5f5d848">https://github.com/vllm-project/vllm/commit/99404f53c72965b41558aceb1bc238087f5f5d848</a>                                         |
| GHSA-fs46-h3ff-q585  | 4,2      | CWE-22   | <a href="https://github.com/saltstack/salt/commit/4b30218edf1a979855ea191d72b30c89f4a5a582">https://github.com/saltstack/salt/commit/4b30218edf1a979855ea191d72b30c89f4a5a582</a>                                                 |
| GHSA-fpwr-67px-3ghx  | 4,3      | CWE-1333 | <a href="https://github.com/huggingface/transformers/commit/92c5ca9dd70de3ade2af2eb835c96215cc50e815">https://github.com/huggingface/transformers/commit/92c5ca9dd70de3ade2af2eb835c96215cc50e815</a>                             |
| GHSA-99pm-ch96-cqp2  | 4,3      | CWE-601  | <a href="https://github.com/dpgaspar/Flask-AppBuilder/commit/32eedbbb5cb483a3e782c5f2732de4a6a650d9b6">https://github.com/dpgaspar/Flask-AppBuilder/commit/32eedbbb5cb483a3e782c5f2732de4a6a650d9b6</a>                           |
| GHSA-x8c6-gj59-6rx8  | 4,3      | CWE-770  | <a href="https://github.com/libp2p/py-libp2p/commit/e150d3153af30530ce61d751bc166e099f1ead7a">https://github.com/libp2p/py-libp2p/commit/e150d3153af30530ce61d751bc166e099f1ead7a</a>                                             |
| GHSA-g93c-p2mw-p23f  | 4,4      | CWE-22   | <a href="https://github.com/dagster-io/dagster/commit/dbb064c2ddda74265b8174edd9775e1302ca6ba0">https://github.com/dagster-io/dagster/commit/dbb064c2ddda74265b8174edd9775e1302ca6ba0</a>                                         |
| GHSA-5gwj-342r-h886  | 4,8      | CWE-20   | <a href="https://github.com/erdogant/pypickle/commit/14b4cae70a40bb4be6723e238f25382d847a1917">https://github.com/erdogant/pypickle/commit/14b4cae70a40bb4be6723e238f25382d847a1917</a>                                           |
| GHSA-v9w6-9hq9-33ch  | 4,8      | CWE-22   | <a href="https://github.com/HKUDS/LightRAG/commit/60777d535b719631680bc5d0969bdef79ca4eaf">https://github.com/HKUDS/LightRAG/commit/60777d535b719631680bc5d0969bdef79ca4eaf</a>                                                   |
| GHSA-7pr5-w74r-jjj7  | 4,8      | CWE-79   | <a href="https://github.com/stephenmcd/mezzanine/commit/898630d8df48cf3ddb8b9942f59168b93216e3f8">https://github.com/stephenmcd/mezzanine/commit/898630d8df48cf3ddb8b9942f59168b93216e3f8</a>                                     |
| GHSA-n84c-4c34-28gf  | 5        | CWE-1112 | <a href="https://github.com/run-llama/llama_index/commit/702e4340623092fac4cf2fe95eb9465034856da3">https://github.com/run-llama/llama_index/commit/702e4340623092fac4cf2fe95eb9465034856da3</a>                                   |
| GHSA-37mw-44qp-f5jm  | 5,3      | CWE-1333 | <a href="https://github.com/huggingface/transformers/commit/ebbe9b12dd75b69f92100db684c47f923e262a93">https://github.com/huggingface/transformers/commit/ebbe9b12dd75b69f92100db684c47f923e262a93</a>                             |
| GHSA-9356-575x-2w9m  | 5,3      | CWE-1333 | <a href="https://github.com/huggingface/transformers/commit/701caef704e356dc19c331cc3f3d5f0ecbcb4720a">https://github.com/huggingface/transformers/commit/701caef704e356dc19c331cc3f3d5f0ecbcb4720a</a>                           |
| GHSA-655q-fx9r-782v  | 5,3      | CWE-184  | <a href="https://github.com/mmlaire314/pickleScan/commit/78ce704227c51f070c0c5fb4466d92c62a7aa3d">https://github.com/mmlaire314/pickleScan/commit/78ce704227c51f070c0c5fb4466d92c62a7aa3d</a>                                     |
| GHSA-gpmg-4x4g-mr5r  | 5,3      | CWE-209  | <a href="https://github.com/ome/omero-web/commit/8aa2789e8f759c73f1517abe9a0abd44e86644ad">https://github.com/ome/omero-web/commit/8aa2789e8f759c73f1517abe9a0abd44e86644ad</a>                                                   |
| GHSA-p74j-wj7f-5x9w  | 5,3      | CWE-440  | <a href="https://github.com/run-llama/llama_index/commit/169e1c0e759228fec4cfaf716e4f951e131de77">https://github.com/run-llama/llama_index/commit/169e1c0e759228fec4cfaf716e4f951e131de77</a>                                     |
| GHSA-9h1g-9r4m-mvj7  | 5,3      | CWE-522  | <a href="https://github.com/psf/requests/commit/96ba401c1296ab1dda74a2365ef36d887f1144ef">https://github.com/psf/requests/commit/96ba401c1296ab1dda74a2365ef36d887f1144ef</a>                                                     |
| GHSA-48p4-8xcl-vxj5  | 5,3      | CWE-601  | <a href="https://github.com/urllib3/urllib3/commit/7eb4a2aaf49a279c29b6d1f0ed0f42e9736194f">https://github.com/urllib3/urllib3/commit/7eb4a2aaf49a279c29b6d1f0ed0f42e9736194f</a>                                                 |
| GHSA-pq67-6m6g-qmj2v | 5,3      | CWE-601  | <a href="https://github.com/urllib3/urllib3/commit/0f5b1329126d5be6de501f9d1e3e36738bc08857">https://github.com/urllib3/urllib3/commit/0f5b1329126d5be6de501f9d1e3e36738bc08857</a>                                               |
| GHSA-8j24-cjrg-gr2m  | 5,3      | CWE-770  | <a href="https://github.com/django/django/commit/9f3419b519799d69f2a67b0b9d25abe2e70d03e0">https://github.com/django/django/commit/9f3419b519799d69f2a67b0b9d25abe2e70d03e0</a>                                                   |
| GHSA-2c2j-9gv5-cj73  | 5,3      | CWE-770  | <a href="https://github.com/encode/starlette/commit/9f7ec2eb512fcc3fe90b43cb9dd9e1d08696bec1">https://github.com/encode/starlette/commit/9f7ec2eb512fcc3fe90b43cb9dd9e1d08696bec1</a>                                             |
| GHSA-gqcv-vg9f-5nr3  | 5,3      | CWE-284  | <a href="https://github.com/BerriAI/litellm/commit/da3ae00bd68f451ed8dd0fbc0a9fd34bde5554d6">https://github.com/BerriAI/litellm/commit/da3ae00bd68f451ed8dd0fbc0a9fd34bde5554d6</a>                                               |
| GHSA-7f3f-x5f5-79gw  | 5,6      | CWE-22   | <a href="https://github.com/saltstack/salt/commit/9445f496fed61b15dc4364818007e5b765b0746f">https://github.com/saltstack/salt/commit/9445f496fed61b15dc4364818007e5b765b0746f</a>                                                 |
| GHSA-989c-m532-p2hv  | 5,6      | CWE-770  | <a href="https://github.com/saltstack/salt/commit/e39116fb87bf4db9bcb9aade8258c66df87d41fe">https://github.com/saltstack/salt/commit/e39116fb87bf4db9bcb9aade8258c66df87d41fe</a>                                                 |
| GHSA-f2f7-gj54-6vvp  | 6,1      | CWE-502  | <a href="https://github.com/huyouga/LLaMA-Factory/commit/2989d39239d2f4fe584c1e1180ba46b9768af2a">https://github.com/huyouga/LLaMA-Factory/commit/2989d39239d2f4fe584c1e1180ba46b9768af2a</a>                                     |
| GHSA-3j8r-jf9w-5cmh  | 6,2      | CWE-22   | <a href="https://github.com/run-llama/llama_index/commit/a86c96ae0e662492eeb471b658ae849a93f628ff">https://github.com/run-llama/llama_index/commit/a86c96ae0e662492eeb471b658ae849a93f628ff</a>                                   |
| GHSA-rm67-4c8j-hjjh  | 6,3      | CWE-200  | <a href="https://github.com/naubobot/naubobot/commit/9c892dc300429948a4714f743c9c2879d8987340">https://github.com/naubobot/naubobot/commit/9c892dc300429948a4714f743c9c2879d8987340</a>                                           |
| GHSA-hx32-3m1r-gjgf  | 6,3      | CWE-22   | <a href="https://github.com/saltstack/salt/commit/f7c28ffb18dbf693a15b1ba9493918de3c88bc3">https://github.com/saltstack/salt/commit/f7c28ffb18dbf693a15b1ba9493918de3c88bc3</a>                                                   |
| GHSA-hw6f-rfj7-jf7j  | 6,3      | CWE-444  | <a href="https://github.com/eventlet/eventlet/commit/0bfbed1117d392559e25b4bfbfc941754de88fb">https://github.com/eventlet/eventlet/commit/0bfbed1117d392559e25b4bfbfc941754de88fb</a>                                             |
| GHSA-9hcf-v7m4-6m2j  | 6,5      | CWE-248  | <a href="https://github.com/vllm-project/vllm/commit/08bf7840780980c57568c573c70a6a8db94fd45f">https://github.com/vllm-project/vllm/commit/08bf7840780980c57568c573c70a6a8db94fd45f</a>                                           |
| GHSA-5h9g-5r78-2ghj  | 6,5      | CWE-440  | <a href="https://github.com/run-llama/llama_index/commit/29b2e07e64ed7d302b1cc058185560b28aa1352">https://github.com/run-llama/llama_index/commit/29b2e07e64ed7d302b1cc058185560b28aa1352</a>                                     |
| GHSA-3wxx-q3gv-pvvv  | 6,5      | CWE-674  | <a href="https://github.com/run-llama/llama_index/commit/c032843a02ce38fd8f284b2aa5a37fd1c17ae635">https://github.com/run-llama/llama_index/commit/c032843a02ce38fd8f284b2aa5a37fd1c17ae635</a>                                   |
| GHSA-hf3c-wxg2-49q9  | 6,5      | CWE-770  | <a href="https://github.com/vllm-project/vllm/commit/cb84e45ac75b42ba6795145923e8eb323bb825ad">https://github.com/vllm-project/vllm/commit/cb84e45ac75b42ba6795145923e8eb323bb825ad</a>                                           |
| GHSA-h7x8-jy97-fvvp  | 6,6      | CWE-22   | <a href="https://github.com/dagster-io/dagster/commit/3a3cec2b51577c4970e6ffc4c199cda6418c09a9d">https://github.com/dagster-io/dagster/commit/3a3cec2b51577c4970e6ffc4c199cda6418c09a9d</a>                                       |
| GHSA-fcr4-h6c4-rvpm  | 6,7      | CWE-77   | <a href="https://github.com/saltstack/salt/commit/9445f496fed61b15dc4364818007e5b765b0746f">https://github.com/saltstack/salt/commit/9445f496fed61b15dc4364818007e5b765b0746f</a>                                                 |
| GHSA-j47q-cr62-w448  | 6,9      | CWE-1333 | <a href="https://github.com/rennf93/fastapi-guard/commit/d9d50e8130b7b434cdc1b001b8cfd03a06729f7f">https://github.com/rennf93/fastapi-guard/commit/d9d50e8130b7b434cdc1b001b8cfd03a06729f7f</a>                                   |
| GHSA-x7q2-wr7g-qxmfl | 6,9      | CWE-208  | <a href="https://github.com/django/django/commit/07cefdee4a9d1fcd9a3a631cbd07b78defd1923b">https://github.com/django/django/commit/07cefdee4a9d1fcd9a3a631cbd07b78defd1923b</a>                                                   |
| GHSA-gmhf-gg9b-jw42  | 6,9      | CWE-208  | <a href="https://github.com/XML-Security/signxml/commit/1b501faacf34cf978a52dbc6915ec11e27611cd">https://github.com/XML-Security/signxml/commit/1b501faacf34cf978a52dbc6915ec11e27611cd</a>                                       |
| GHSA-p7a8-grj9-3m8w  | 6,9      | CWE-22   | <a href="https://github.com/copier-org/copier/commit/fdbc0167cc22780b497e4db176feaf6024757d6">https://github.com/copier-org/copier/commit/fdbc0167cc22780b497e4db176feaf6024757d6</a>                                             |
| GHSA-6vx8-pcww-xhf4  | 6,9      | CWE-303  | <a href="https://github.com/XML-Security/signxml/commit/e3c0c2b82a3329a65d91783065764d9c98b8c7600">https://github.com/XML-Security/signxml/commit/e3c0c2b82a3329a65d91783065764d9c98b8c7600</a>                                   |
| GHSA-jh75-99hh-qvx9  | 6,9      | CWE-400  | <a href="https://github.com/django/django/commit/27900fe56f33cabb4aeb6ccb82f92bab29073a8">https://github.com/django/django/commit/27900fe56f33cabb4aeb6ccb82f92bab29073a8</a>                                                     |
| GHSA-mwfg-948f-2cc5  | 6,9      | CWE-79   | <a href="https://github.com/MobSF/Mobile-Security-Framework-MobSF/commit/6987a946485a795f4fd38cebbdb4860b368a1995d">https://github.com/MobSF/Mobile-Security-Framework-MobSF/commit/6987a946485a795f4fd38cebbdb4860b368a1995d</a> |
| GHSA-847f-9342-285h  | 6,9      | CWE-93   | <a href="https://github.com/python-hyper/h2/commit/035e9899f95e3709af098f578bfc3cd302298e3a">https://github.com/python-hyper/h2/commit/035e9899f95e3709af098f578bfc3cd302298e3a</a>                                               |
| GHSA-hwpg-x5hw-vpv9  | 7        | CWE-79   | <a href="https://github.com/dgltmoon/changedetection.io/commit/3d5a544ae674cfce517adcd498877a8d760d0931">https://github.com/dgltmoon/changedetection.io/commit/3d5a544ae674cfce517adcd498877a8d760d0931</a>                       |
| GHSA-v56r-hwv5-mxg6  | 7,1      | CWE-20   | <a href="https://github.com/element-hq/synapse/commit/2277df2a1eb685f85040ef98fa21d441aa4cd389">https://github.com/element-hq/synapse/commit/2277df2a1eb685f85040ef98fa21d441aa4cd389</a>                                         |
| GHSA-xqpg-92fq-grfg  | 7,5      | CWE-22   | <a href="https://github.com/pyload/pyload/commit/fc4b136e9c4e7dcb8be467ae802cb2c370a71b0">https://github.com/pyload/pyload/commit/fc4b136e9c4e7dcb8be467ae802cb2c370a71b0</a>                                                     |
| GHSA-fmrf-6vj9-qc7f  | 7,5      | CWE-22   | <a href="https://github.com/run-llama/llama_index/commit/266be3b3a61f158112726d75a5f5f0b9e34ded0">https://github.com/run-llama/llama_index/commit/266be3b3a61f158112726d75a5f5f0b9e34ded0</a>                                     |
| GHSA-2rhq-96q8-4vyq  | 7,5      | CWE-29   | <a href="https://github.com/run-llama/llama_index/commit/cdeaab91a204d1c3527f177dac37390327aef274">https://github.com/run-llama/llama_index/commit/cdeaab91a204d1c3527f177dac37390327aef274</a>                                   |
| GHSA-rcx4-3w6r-4v47  | 7,5      | CWE-400  | <a href="https://github.com/vllm-project/vllm/commit/d8b736f913a59117803d6701521d2e4861701944">https://github.com/vllm-project/vllm/commit/d8b736f913a59117803d6701521d2e4861701944</a>                                           |
| GHSA-7c85-87cp-mr6g  | 7,5      | CWE-400  | <a href="https://github.com/run-llama/llama_index/commit/3c65db2947271de3bd1927dc66a044da385de4da">https://github.com/run-llama/llama_index/commit/3c65db2947271de3bd1927dc66a044da385de4da</a>                                   |
| GHSA-8w49-h7vm-mf3c  | 7,5      | CWE-770  | <a href="https://github.com/tornadoweb/tornado/commit/d5ba4a1b95fbf7c6a3e54313262639b198291533">https://github.com/tornadoweb/tornado/commit/d5ba4a1b95fbf7c6a3e54313262639b198291533</a>                                         |
| GHSA-jf8f-2vnm-885j  | 7,5      | CWE-770  | <a href="https://github.com/vllm-project/vllm/commit/a0304dc504c85f4d1328ef47c6483046a13641c">https://github.com/vllm-project/vllm/commit/a0304dc504c85f4d1328ef47c6483046a13641c</a>                                             |
| GHSA-w42r-rfj7-c633  | 7,5      | CWE-776  | <a href="https://github.com/run-llama/llama_index/commit/4f6ee062b19212106a2632af9c9521cf70a3584">https://github.com/run-llama/llama_index/commit/4f6ee062b19212106a2632af9c9521cf70a3584</a>                                     |
| GHSA-8jhr-wpcm-hh4h  | 7,6      | CWE-79   | <a href="https://github.com/HumanSignal/label-studio/commit/97db9e7b16783e1f6052eb432a6f014f80ef268d">https://github.com/HumanSignal/label-studio/commit/97db9e7b16783e1f6052eb432a6f014f80ef268d</a>                             |
| GHSA-2gxp-6r36-m97r  | 7,6      | CWE-79   | <a href="https://github.com/zmievska/cadwyn/commit/b424ecd57cd8dabb8cfe39b8f8ccafea629c7728">https://github.com/zmievska/cadwyn/commit/b424ecd57cd8dabb8cfe39b8f8ccafea629c7728</a>                                               |
| GHSA-5rjg-fvgr-3xxf  | 7,7      | CWE-22   | <a href="https://github.com/pypa/setuptools/commit/250a6d17978f9f6ac3ac887091f2d32886fb0b0b">https://github.com/pypa/setuptools/commit/250a6d17978f9f6ac3ac887091f2d32886fb0b0b</a>                                               |
| GHSA-rf6p-pxhg-684g  | 7,8      | CWE-20   | <a href="https://github.com/rennf93/fastapi-guard/commit/0829292c322d33dc14ab00c5451c5c138148035a">https://github.com/rennf93/fastapi-guard/commit/0829292c322d33dc14ab00c5451c5c138148035a</a>                                   |
| GHSA-pw95-88fq-3j6f  | 7,8      | CWE-611  | <a href="https://github.com/langroid/langroid/commit/36e7e7db4dd1636de225c2c66c84052b1e9ac3c3">https://github.com/langroid/langroid/commit/36e7e7db4dd1636de225c2c66c84052b1e9ac3c3</a>                                           |
| GHSA-g99h-56mw-8263  | 7,8      | CWE-78   | <a href="https://github.com/run-llama/llama_index/commit/b57e6738c53ca82d88658b82f2f82d21c7839c7d">https://github.com/run-llama/llama_index/commit/b57e6738c53ca82d88658b82f2f82d21c7839c7d</a>                                   |
| GHSA-pwh4-6r3m-j2rf  | 7,8      | CWE-89   | <a href="https://github.com/pyload/pyload/commit/134edcd6fe2a10c393743c254da3c9d90b74258f">https://github.com/pyload/pyload/commit/134edcd6fe2a10c393743c254da3c9d90b74258f</a>                                                   |
| GHSA-c46w-gr7f-jm2p  | 8,1      | CWE-285  | <a href="https://github.com/saltstack/salt/commit/41d834bf800d86fc496e4fac23875fc2aca7c62">https://github.com/saltstack/salt/commit/41d834bf800d86fc496e4fac23875fc2aca7c62</a>                                                   |
| GHSA-jf4f-3j3r-78r9  | 8,1      | CWE-285  | <a href="https://github.com/berriai/litellm/commit/566d9354aab4215091b2e51ad0333e948125fa1b">https://github.com/berriai/litellm/commit/566d9354aab4215091b2e51ad0333e948125fa1b</a>                                               |
| GHSA-f3mf-hm6v-jfhh  | 8,1      | CWE-915  | <a href="https://github.com/mesop-dev/mesop/commit/748e20d4a363d89b841d62213f5b0c6b4bed788f">https://github.com/mesop-dev/mesop/commit/748e20d4a363d89b841d62213f5b0c6b4bed788f</a>                                               |
| GHSA-8qym-5x2c-j2w7  | 8,2      | CWE-674  | <a href="https://github.com/protocolbuffers/protobuf/commit/17838bda2943d08b8a9d4df5b68f5f04f26d901">https://github.com/protocolbuffers/protobuf/commit/17838bda2943d08b8a9d4df5b68f5f04f26d901</a>                               |
| GHSA-wjrh-hj83-3wh7  | 8,2      | CWE-918  | <a href="https://github.com/codingjoe/django-select2/commit/e5f41e6edba004d35f94915ff5e2559f4853412">https://github.com/codingjoe/django-select2/commit/e5f41e6edba004d35f94915ff5e2559f4853412</a>                               |
| GHSA-pwhh-q4h6-w599  | 8,4      | CWE-276  | <a href="https://github.com/spotipy-dev/spotipy/commit/1ca453f6ef87a2a9e9876f52b6cb38d13532ccf2">https://github.com/spotipy-dev/spotipy/commit/1ca453f6ef87a2a9e9876f52b6cb38d13532ccf2</a>                                       |
| GHSA-378x-6p4f-8jgm  | 8,4      | CWE-502  | <a href="https://github.com/skops-dev/skops/commit/29d61ea8a92f2bde6830e8f32cc27a1a87211cda">https://github.com/skops-dev/skops/commit/29d61ea8a92f2bde6830e8f32cc27a1a87211cda</a>                                               |
| GHSA-h5gc-rm8j-5gpr  | 8,4      | CWE-918  | <a href="https://github.com/langchain-ai/langchain/commit/e188d4ecb085d4561a0be3c583d26aa9c2c3283f">https://github.com/langchain-ai/langchain/commit/e188d4ecb085d4561a0be3c583d26aa9c2c3283f</a>                                 |
| GHSA-3xw7-v6cj-5q8h  | 8,5      | CWE-22   | <a href="https://github.com/copier-org/copier/commit/3feea3b3ff3c20d80cbb16a2f3b9567ffc506ed1">https://github.com/copier-org/copier/commit/3feea3b3ff3c20d80cbb16a2f3b9567ffc506ed1</a>                                           |
| GHSA-7753-xrfw-ch36  | 8,6      | CWE-674  | <a href="https://github.com/run-llama/llama_index/commit/c032843a02ce38fd8f284b2aa5a37fd1c17ae635">https://github.com/run-llama/llama_index/commit/c032843a02ce38fd8f284b2aa5a37fd1c17ae635</a>                                   |
| GHSA-cfmv-h8fx-85m7  | 8,7      | CWE-22   | <a href="https://github.com/ietf-tools/xml2rfc/commit/f2b245bc0aeaac0667c8f74e976c466c5991f0e4">https://github.com/ietf-tools/xml2rfc/commit/f2b245bc0aeaac0667c8f74e976c466c5991f0e4</a> </                                      |

## .2. Results

| ID                  | LOC  | Files | CC | Blocks | Edges | Paths | Structural Churn Score |
|---------------------|------|-------|----|--------|-------|-------|------------------------|
| GHSA-9548-qrrj-x5pj | 72   | 2     | 0  | -2     | -4    | -620  | 116,667                |
| GHSA-g92j-qhmh-64v2 | 6    | 1     | 0  | 0      | 0     | 0     | 1,167                  |
| GHSA-v7vm-rhmg-8j2r | 43   | 2     | 7  | 0      | 6     | 3     | 10,167                 |
| GHSA-4qqf-9m5c-w2c5 | 14   | 2     | 3  | 0      | 5     | 2     | 4,333                  |
| GHSA-7j9p-67mm-5g87 | 26   | 2     | 1  | 2      | 3     | 1     | 5,833                  |
| GHSA-fm6c-f59h-7mmg | 2    | 1     | 0  | 0      | 0     | 0     | 0,5                    |
| GHSA-4qjh-9fv9-r85r | 258  | 10    | 14 | 1      | 22    | 12    | 52,833                 |
| GHSA-hhfg-fwrw-87w7 | 22   | 2     | 3  | 2      | 3     | 4     | 6                      |
| GHSA-7mpr-5m44-h73r | 3    | 1     | 0  | 0      | 0     | 0     | 0,667                  |
| GHSA-674p-xv2x-rf3g | 2    | 1     | 0  | 0      | 0     | 0     | 0,5                    |
| GHSA-5xh2-23cc-5jc6 | 79   | 5     | 10 | 5      | 26    | 11    | 22,667                 |
| GHSA-c65p-x677-fgj6 | 32   | 1     | 2  | 0      | 2     | -3    | 6,667                  |
| GHSA-r546-h3ff-q585 | 66   | 4     | 4  | 0      | 10    | 26    | 18,333                 |
| GHSA-fpwr-67px-3qhx | 81   | 3     | 3  | 0      | 1     | 2     | 16                     |
| GHSA-99pm-ch96-ccp2 | 2    | 1     | 0  | 0      | 0     | 0     | 0,5                    |
| GHSA-x8c6-gj59-6rx8 | 26   | 1     | 3  | 0      | 7     | 3     | 6,667                  |
| GHSA-q93c-p2mw-p23f | 6    | 1     | 1  | 1      | 3     | 4     | 2,667                  |
| GHSA-5qwj-342r-h886 | 317  | 2     | 16 | 7      | 29    | 16    | 64,5                   |
| GHSA-v9w6-9hq9-33ch | 54   | 1     | 9  | 0      | 13    | 5     | 13,667                 |
| GHSA-7pr5-w74r-jjj7 | 3    | 1     | 0  | 0      | 0     | 0     | 0,667                  |
| GHSA-m84c-4c34-28gf | 11   | 1     | 0  | 0      | 0     | 0     | 2                      |
| GHSA-37mw-44qp-f5jm | 12   | 1     | 1  | 2      | 3     | 4     | 3,833                  |
| GHSA-9356-575x-2w9m | 3    | 1     | 3  | 0      | 3     | 2     | 2                      |
| GHSA-655q-fx9r-782v | 1    | 1     | 0  | 0      | 0     | 0     | 0,333                  |
| GHSA-gpmg-4x4g-mr5r | 9    | 1     | -1 | 0      | -2    | -2    | 2,5                    |
| GHSA-p7j4-jwjf-5x9w | 15   | 1     | 0  | 0      | 0     | 0     | 2,667                  |
| GHSA-9hjh-9r4m-mvj7 | 8    | 1     | -1 | 0      | -3    | -28   | 6,833                  |
| GHSA-48p4-8xcf-vxj5 | 20   | 1     | 4  | 2      | 5     | 8     | 6,667                  |
| GHSA-pq67-6m6q-mj2v | 19   | 2     | 0  | 0      | 5     | 2     | 4,667                  |
| GHSA-8j24-cjrq-gr2m | 6    | 1     | 2  | 1      | 6     | 1     | 2,833                  |
| GHSA-2c2j-9gv5-cj73 | 22   | 1     | 3  | 0      | 5     | 2     | 5,5                    |
| GHSA-qqcw-vg9f-5rr3 | 2451 | 5     | 6  | 15     | 1     | 1     | 413,167                |
| GHSA-7f3f-x5f5-79gw | 78   | 4     | 10 | 0      | 21    | 10    | 20,5                   |
| GHSA-989c-m532-p2hv | 8    | 1     | 0  | 0      | 0     | 0     | 1,5                    |
| GHSA-f2f7-gj54-6vpv | 2    | 1     | 0  | 0      | 0     | 0     | 0,5                    |
| GHSA-3j8r-jf9w-5cmh | 17   | 1     | 2  | 2      | 5     | 2     | 4,833                  |
| GHSA-rh67-4c8j-hjjh | 25   | 2     | 0  | 0      | 8     | 3     | 6,333                  |
| GHSA-xh32-3m67-qjgf | 20   | 2     | 1  | 0      | 4     | 1     | 4,667                  |
| GHSA-hw6f-rjff-j7j7 | 8    | 1     | 4  | 0      | 6     | 2     | 3,5                    |
| GHSA-9hcf-v7m4-6m2j | 10   | 2     | 1  | 1      | 3     | 18    | 5,833                  |
| GHSA-5hq9-5r78-2gjh | 5    | 1     | 0  | 0      | 0     | 0     | 1                      |
| GHSA-3wxx-q3gv-pvvv | 97   | 1     | 1  | 1      | 0     | 0     | 16,667                 |
| GHSA-hf3c-wxg2-49q9 | 23   | 3     | 0  | 0      | 0     | 0     | 4,333                  |
| GHSA-h7x8-jv97-fvvm | 11   | 1     | 1  | 0      | 3     | 1     | 2,833                  |
| GHSA-fcr4-h6c4-rvvp | 78   | 4     | 10 | 0      | 21    | 10    | 20,5                   |
| GHSA-j47q-rc62-w448 | 156  | 3     | 0  | 0      | 0     | 0     | 26,5                   |
| GHSA-x7q2-wr7g-xqmf | 10   | 1     | 0  | 2      | 0     | 2     | 2,5                    |
| GHSA-gmhf-gg8w-jw42 | 5    | 1     | 0  | 0      | 64    |       | 11,667                 |
| GHSA-p7q8-grrj-3m8w | 23   | 2     | 1  | 0      | 5     | 2     | 5,5                    |
| GHSA-6vx8-pcwv-xhf4 | 58   | 1     | 7  | 7      | 13    | 5059  | 857,5                  |
| GHSA-jh75-99hh-qvx9 | 13   | 1     | 2  | 5      | 7     | 70    | 16,333                 |
| GHSA-mwfg-948f-2cc5 | 103  | 6     | 8  | 5      | 22    | 46    | 31,667                 |

| ID                  | LOC | Files | CC  | Blocks | Edges | Paths | Structural Churn Score |
|---------------------|-----|-------|-----|--------|-------|-------|------------------------|
| GHSA-847f-9342-265h | 41  | 1     | 6   | 1      | 6     | 1     | 9,333                  |
| GHSA-hwpg-x5hw-vpv9 | 15  | 2     | 1   | 0      | 2     | 1     | 3,5                    |
| GHSA-v56r-hwv5-mxg6 | 90  | 7     | -1  | 1      | 12    | 4     | 19,167                 |
| GHSA-xqpg-92fq-grfg | 4   | 1     | 0   | 0      | 0     | 0     | 0,833                  |
| GHSA-fmrf-6jv9-qjc7 | 73  | 1     | 3   | 7      | 10    | 3     | 16,167                 |
| GHSA-2rhq-96q8-4vjq | 26  | 1     | 6   | 0      | 10    | 4     | 7,833                  |
| GHSA-rxc4-3w6r-4v47 | 41  | 4     | 2   | 2      | 1     | 1     | 8,5                    |
| GHSA-7c85-87cp-mr6g | 15  | 1     | 1   | 2      | 3     | 1     | 3,833                  |
| GHSA-8w49-h785-mj3c | 38  | 1     | -6  | 0      | -11   | -7    | 10,5                   |
| GHSA-9f8f-2vmf-885j | 2   | 1     | 0   | 0      | 0     | 0     | 0,5                    |
| GHSA-w42r-mrx7-c633 | 35  | 3     | 0   | 0      | 0     | 0     | 6,333                  |
| GHSA-8jhr-wpcm-hh4h | 41  | 2     | -7  | -6     | -19   | -7    | 13,667                 |
| GHSA-2gxp-6r36-m97r | 5   | 1     | 0   | 0      | 0     | 0     | 1                      |
| GHSA-5rjg-fvgr-3xxf | 18  | 1     | 1   | 0      | 3     | 4     | 4,5                    |
| GHSA-rrf6-pxg8-684g | 148 | 4     | 4   | 0      | 0     | 0     | 26                     |
| GHSA-pw95-88fg-3j6f | 18  | 2     | 0   | 0      | 0     | 0     | 3,333                  |
| GHSA-g99h-56mw-8263 | 10  | 1     | 1   | 0      | 0     | 0     | 2                      |
| GHSA-pwh4-6r3m-j2rf | 10  | 1     | 0   | 0      | -4    | -1    | 2,667                  |
| GHSA-c46w-gr7f-jm2p | 24  | 1     | 4   | 0      | 11    | 4     | 7,333                  |
| GHSA-fjcf-3j3r-78rp | 23  | 2     | 1   | 0      | 0     | 0     | 4,333                  |
| GHSA-f3mf-hm6v-jfhh | 4   | 1     | 2   | 0      | 3     | 1     | 1,833                  |
| GHSA-8qvm-5x2c-j2w7 | 31  | 1     | 6   | 1      | 17    | 12    | 11,333                 |
| GHSA-wjrh-hj83-3wh7 | 4   | 1     | 0   | 0      | 0     | 0     | 0,833                  |
| GHSA-pwhh-q4h6-w599 | 4   | 1     | 0   | 0      | 2     | 1     | 1,333                  |
| GHSA-378x-6p4f-8jgm | 85  | 1     | 4   | 3      | 10    | 4     | 10,167                 |
| GHSA-h5gc-rm8j-5gpr | 138 | 3     | 5   | 7      | 11    | 5     | 28,167                 |
| GHSA-3xw7-v6cj-5q8h | 21  | 1     | 2   | 0      | 8     | 3     | 5,833                  |
| GHSA-7753-xrfw-ch36 | 97  | 1     | 1   | 1      | 0     | 0     | 16,667                 |
| GHSA-cfmv-h8fx-85m7 | 14  | 2     | 2   | 0      | 5     | 2     | 4,167                  |
| GHSA-3qhf-m339-9g5v | 59  | 1     | 1   | 0      | 0     | 0     | 10,167                 |
| GHSA-j975-95f5-7wqh | 183 | 1     | 4   | 0      | 0     | 0     | 31,333                 |
| GHSA-4v6w-xpmh-gfgp | 45  | 3     | 2   | 0      | 8     | 3     | 10,167                 |
| GHSA-79j6-g2m3-jgfw | 671 | 2     | 129 | 104    | 246   | 5848  | 1166,667               |
| GHSA-c9rc-mg46-23w3 | 77  | 7     | 6   | 0      | 14    | 2     | 17,667                 |
| GHSA-9gvj-pp9x-gcfr | 5   | 1     | 0   | 0      | 0     | 0     | 1                      |
| GHSA-2689-cw26-6cpj | 50  | 1     | 0   | -1     | -4    | -1    | 9,5                    |
| GHSA-pv4p-cwwg-4rph | 9   | 2     | 1   | 0      | 4     | 1     | 2,833                  |
| GHSA-rvqx-wpfb-mfx7 | 3   | 1     | 0   | 0      | 0     | 0     | 0,667                  |
| GHSA-8pcp-r83j-fc92 | 7   | 1     | 0   | 0      | 0     | 0     | 1,333                  |
| GHSA-48rp-jc79-2264 | 9   | 1     | 1   | -1     | 3     | 1     | 2,667                  |
| GHSA-hjq4-87xh-g4fv | 22  | 1     | 1   | 0      | 4     | 1     | 4,833                  |
| GHSA-33xw-247w-6hmc | 9   | 1     | 2   | 0      | 0     | 0     | 2                      |
| GHSA-339r-cjv9-x78g | 34  | 1     | 0   | -1     | -3    | -1    | 6,667                  |
| GHSA-v3c8-3pr6-gr7p | 9   | 2     | 1   | 0      | 0     | 0     | 2                      |
| GHSA-jqq5-wc57-f8hj | 317 | 3     | 37  | 9      | 83    | 38    | 81,167                 |
| GHSA-mrmq-3q62-6cc8 | 59  | 2     | 15  | 7      | 14    | 5     | 17                     |
| GHSA-6v92-r5mx-h5fx | 123 | 1     | 4   | 0      | 5     | 3     | 22,667                 |
| GHSA-hj4w-hm2g-p6w5 | 21  | 1     | 0   | 0      | 0     | 0     | 3,667                  |