AW4C: A Commit-Aware C Dataset for Actionable Warning Identification

Zhipeng Liu[1], Meng Yan[1], Zhipeng Gao[2], Dong Li[1], Xiaohong Zhang[1], Dan Yang[1]
{zhipengliu,mengy,lidong,xhongz,dyang}@cqu.edu.cn,zhipeng.gao@zju.edu.cn
[1]School of Big Data and Software Engineering, Chongqing University, China
[2]Shanghai Institute for Advanced Study, Zhejiang University, China

Abstract

Excessive non-actionable warnings generated by static program analysis tools can hinder developers from utilizing these tools effectively. Leveraging learning-based approaches for actionable warning identification has demonstrated promise in boosting developer productivity, minimizing the risk of bugs, and reducing code smells. However, the small sizes of existing datasets have limited the model choices for machine learning researchers, and the lack of aligned fix commits limits the scope of the dataset for research. In this paper, we present AW4C, an actionable warning C dataset that contains 38,134 actionable warnings mined from more than 500 repositories on GitHub. These warnings are generated via Cppcheck, and most importantly, each warning is precisely mapped to the commit where the corrective action occurred. To the best of our knowledge, this is the largest publicly available actionable warning dataset for C programming language to date. The dataset is suited for use in machine/deep learning models and can support a wide range of tasks, such as actionable warning identification and vulnerability detection. Furthermore, we have released our dataset[1] and a general framework for collecting actionable warnings on GitHub[2] to facilitate other researchers to replicate our work and validate their innovative ideas.

CCS Concepts

• **Software and its engineering** → **Software maintenance tools**; *Software creation and management*; *Software testing and debugging*; • **Mathematics of computing** → *Data mining*.

Keywords

Static program analysis, Actionable warning identification

ACM Reference Format:
Zhipeng Liu[1], Meng Yan[1], Zhipeng Gao[2], Dong Li[1], Xiaohong Zhang[1], Dan Yang[1]. 2024. AW4C: A Commit-Aware C Dataset for Actionable Warning Identification. In *In 21th International Conference on Mining Software Repositories (MSR '24), April 14–15, 2024.* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

In recent years, Static Program Analysis (SPA) has been widely utilized by developers in software development, especially among large companies like Facebook[3]. Developers have integrated static program analysis tools (e.g., SpotBug, Cppcheck, FindBugs, Infer) into their workflow to ensure the robustness and reliability of applications [2]. The goal of these tools is to detect potential bugs that may affect the correctness, security, and performance of software applications. However, a primary challenge in adopting static program analysis tools in practice is their high false alarm rates [2, 12]. To fill the gap and help developers better utilize static program analysis tools, researchers have proposed automated actionable warning identification techniques, where "actionable warning" refers to important anomalies and fixable warnings that need to be acted by developers [7].

Recently, researchers have proposed various learning-based actionable warning identification approaches. For example, Hanam et al. [6] achieved binary classification by finding warnings with similar patterns, where the patterns were determined based on the codes surrounding the warning. Machine learning has also been used to take semantic and syntactic features during pattern recognition. Other studies using machine learning to recognize warnings are numerous [5, 8, 15, 16]. However, all of the aforementioned machine learning-based approaches heavily rely on the quality and quantity of the actionable warning dataset. Wang et al. [14] and Yang et al. [15] introduced a dataset derived from Findbugs consisting of 768 actionable warnings. Nonetheless, this dataset exhibits several limitations:

(1) The small size of the dataset and the limited traceability of warning fixes restrict the dataset's application. Moreover, its reliance on complex feature engineering hampers its application in end-to-end deep learning.
(2) The construction process, not automated and difficult to generalize, is limited to specific programming languages and static analysis tools.

To address the aforementioned problems, we introduce a dataset called AW4C and a corresponding dataset construction framework. The contributions of this paper are:

- **Dataset.** We have collected and published a large C actionable warning dataset, AW4C, which contains traceable warning sources, code fix patches, and warning attributes. To the best of our knowledge, this is by far the largest actionable warning dataset for C programming language.
- **Data Mining Framework.** We propose a commit-aware framework for automatic actionable warning mining. Our

[1] https://zenodo.org/doi/10.5281/zenodo.10277281
[2] https://github.com/LiuZhipeng99/AW4C

[3] https://github.com/facebook/infer

MSR 2024, April 2024, Lisbon, Portugal
© 2024 Association for Computing Machinery.
ACM ISBN 978-1-XXXX-XXXX-X/23/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

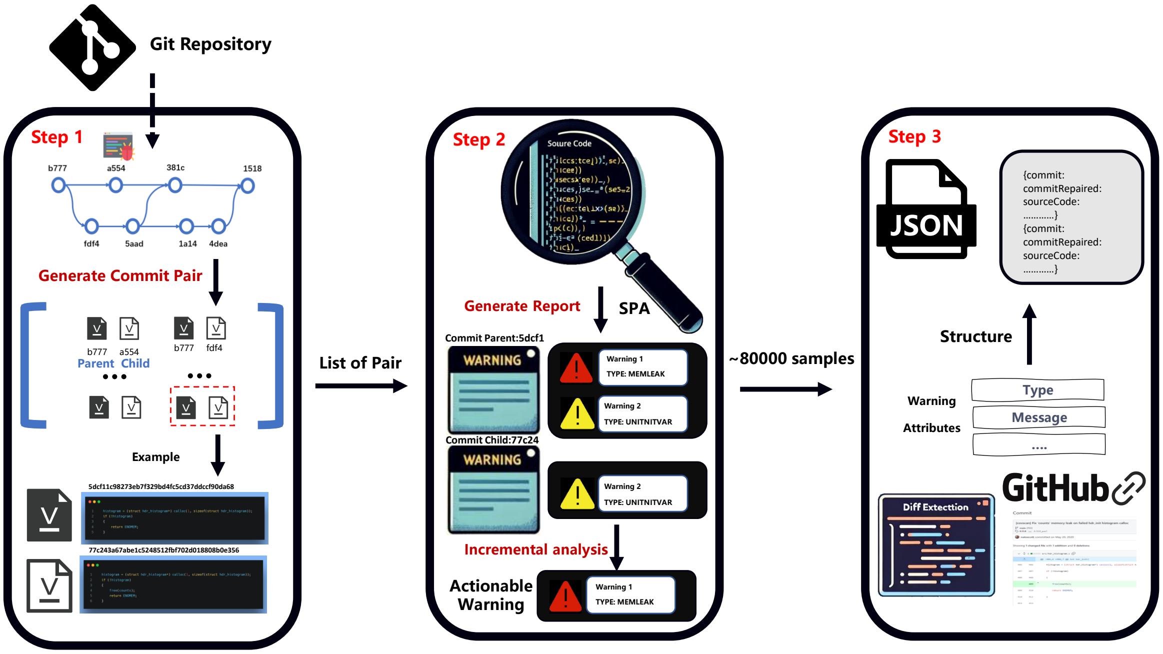

Figure 1: Overview of dataset construction methods.

framework can be easily applied to other static program analysis tools and generalized to other programming languages.

2 CONSTRUCTION OF THE DATASET

We collected C repositories based on the popularity of the project (i.e., the number of stars of the GitHub repository) and selected over 500 top repositories. Since our approach is a commit-aware approach, we filtered out projects with an excessively large number of commits (i.e., more than 100K commits) due to the timely complexity of SPA for each commit. We selected Cppcheck, a well-known tool for static program analysis, which is specifically designed for the C and C++ programming languages and boasts a comprehensive set of scanning rules.

An overview of our proposed framework is shown in Figure 1. The following sections illustrate the sample collection steps and how these samples were formatted to achieve our goals.

2.1 Mining of Warning-Fixing Commit Pairs

To collect actionable warnings from GitHub repository commit histories, we adhered to the previously defined criteria for actionable warnings. Specifically, a warning identified in one revision that disappears in a subsequent revision is considered actionable within our framework. We constructed AW4C in the following three steps:

Step 1: We first parse the commit history of each GitHub repository into a list of commit pairs. In this context, a commit pair refers to the pair of the current commit and its immediate predecessor, known as the parent commit. Commit pairs are used to get the code before and after possible warning fixes. Due to the presence of multiple branches in development, the commit history in the repository is not linear. By using the Git CLI (Command Line Interface), we can extract a time series of commits, as illustrated by the list [b777,fdf4,a554,5aad,...] in Figure 1. For each repository, we systematically traverse this list of commits. During this process, we identify the immediate predecessor of each commit, thereby forming pairs consisting of a commit and its parent. Merge commits are excluded from this pairing process because the actual code changes typically occur in the parent commits of these merges.

Step 2: We use Cppcheck to analyze a commit pair, generate two separate lists of warnings for a commit pair, and get actionable warnings after comparing. If we can get the introduced warnings and the disappeared warnings, then we mark the disappearing warnings as actionable warnings. The verification of this labeling is further discussed in Section 5. In this step, we design the incremental analysis method for the warning reports and use it on Cppcheck. The second box in Figure 1 illustrates the two warning lists for a given commit pair. These warnings, initially, do not exhibit a direct correlation between the two reports. However, through the application of incremental warning analysis, actionable warnings can be identified and extracted from these lists. A comprehensive explanation of this algorithm's design is provided in Section 2.2.

Step 3: After obtaining the actionable warnings, we structure the dataset for a succinct overview, capturing warning attributes through XML parsing of the Cppcheck report, and including commit-level details for traceability. These two types of information are sufficient to trace other features of the warnings, and we try to trace some code-related features in Section 2.3 for future research.

2.2 Analysis of Incremental Warnings

Incremental warning analysis is used to obtain fixed warnings and newly introduced warnings from the commit pair's warning report. A major challenge lies in verifying if two warnings reported in different commits are the same warning. Typically a warning in a report consists of several attributes, such as type, message, and location of the warning in the code. However, none of these attributes alone can uniquely identify a warning's source. Two different warnings may have the same type and message, and the same warning may have a different file name or line number change due to a file modification.

To effectively tackle this challenge, we propose assigning a unique identifier (ID) to each warning. This ID is generated by computing a hash from the quaternion:

$$(\text{severity}, \text{type}, \text{message}, \text{filename}, \text{additional}).$$

The "filename" is specifically adjusted through filtering or replacing to handle file deletions and renames across commits. The "message" is derived by extracting location details via string processing. In addition, the importance of the "additional" field varies between SPA tools, such as the important "Info" field in Cppcheck. This algorithmic approach ensures accurate tracking of warning alterations, thereby enhancing the reliability of our analysis.

Furthermore, to prevent assigning the same ID to warnings from different sources, we use Natural Language Processing (NLP) techniques to ensure the accuracy of the incremental analysis of warnings. This evaluates the textual similarity of warnings assigned to the same ID. Future research will explore additional methodologies, such as code similarity and code structure differences [4].

2.3 Tracing Warning-Fixing

In addition to providing an actionable warning dataset, for each actionable warning in our dataset, we also provide its tracing information. we can retrieve the versions of the code before and after the commit, either locally or from GitHub. During code extraction, we observed that warnings are typically at the line level, making the extraction of contextual code challenging. The solution we implemented is to extract the changes near the warning line from the git diff, which can be obtained from the Git CLI and parsed.

3 DATASET DESCRIPTION AND STATISTICS

The provenance of AW4C ensures its reliability and wide applicability, as the repositories were chosen from the most popular and actively developed open source C projects on GitHub. We organized the list of repositories through CSV files and designed our data collection tools to regenerate consistent results under the same conditions, thus ensuring reproducibility. Each repository generated two JSON files, one containing non-actionable warnings and the other containing actionable warnings. Non-actionable warnings refer to those that were never fixed since introduction. Our final released dataset consists of a single merged and compressed JSON file containing a total of 38,134 actionable warnings and 47,506 non-actionable warnings from the selected repositories. The dataset and associated tools are available on GitHub. Details on usage and reproduction can be found in the Readme file. Table 1 describes the dataset structure and features, including categories, names, and descriptions. The warning attributes characterize warnings and apply to other static analysis tools. As our mining approach is commit-aware, each warning-commit pair contains commit-related information.

Table 1: Description of Warning Features.

	Features	Description
Warning Attribute	SPA Tool	Types of static program analysis tools, such as Cppcheck.
	Message	Detailed text description of the warning.
	Type	Type of warning, such as memleak, nullpointer.
	Severity	This field indicates the seriousness of the warning.
	CWE	The Common Weakness Enumeration ID of the warning.
	FilePath	Path to the file that caused the warning.
	LineNumber	The line number where the warning is located.
	Id	ID of the warning generated by the incremental analysis.
Commit Detail	Message	Short description information of the commit.
	Hash	The commit hash of parent in the commit pair.
	ChildHash	The commit hash of the child in the commit pair
Warning Trace	Link	Commit link where fixes or introductions have occurred.
	Context	Context code for warnings.
	DiffText	Code diff text between the two commits.

Table 2 shows 12 prominent C repositories included in our dataset, along with some key statistics. The "Number of Commits" column refers to the number of versions/commits traced during the dataset mining process. The full scope of commits was not mined as we implemented selective filtering to reduce mining time for large projects. Specifically, for some repositories in Table 2, the commits are before a specific Tag: Redis up to Tag 1.3.6, cURL up to Tag curl-7_16_0, Vim up to Tag v7.0.030, and MPV up to Tag v0.1.0.

Figure 2 shows the distribution of warning and action rates across the studied repositories. An important observation is that projects exhibiting high action rates tend to have higher levels of completion, while some inactive or early-stage projects like wrk and the initial development of vim recorded lower action rates. In

Table 2: Repositories included in the dataset.

Repository	# Commits	# Actionable	# Warnings
Redis	1175	149	287
cURL	8154	330	531
Vim	904	268	2390
MPV	3448	604	1165
scrcpy	2261	98	236
tmux	10147	674	984
stb	2154	19	49
Darknet	455	923	1218
masscan	855	240	430
libuv	5372	542	1110
jq	1636	106	304
wrk	80	8	553
others	-	34173	76446
all	-	**38134**	**85640**

addition to the insights in Figure 2, deeper analysis of the dataset promises to uncover more valuable findings.

4 DATASET USAGE

Our AW4C dataset supports a general range of machine learning applications related to warnings and vulnerabilities research.

Actionable warnings identification and prediction. Our dataset can be used to build a machine learning model to identify actionable warnings and to reduce false positives in static analysis tools [1, 9]. This helps developers to locate potential security issues more quickly, improving the efficiency and quality of software development. In practice, the model can be integrated into Continuous Integration and Continuous Delivery (CI/CD) processes to automate the filtering of actionable warnings. In this way, development teams can get immediate feedback on actionable warnings as soon as the code is committed to the repository. In addition, we can evaluate the consistency and accuracy between different tools, and this comparison helps to select the most appropriate tool for a specific project, improving the security of the development ecosystem.

Auto-patching and vulnerabilities detection. The actionable warnings in our dataset include warning types. The source code of the dataset is also traceable, allowing researchers to model different levels of code vulnerabilities, such as file, function, and line level, for vulnerability detection [3, 10, 11].

Code in open source projects is often heavily modified by contributors, and an effective patch detection mechanism allows developers to be well informed of any hidden security patches [18]. Our dataset can also serve as a fix patch dataset, containing common bug-fix strategies and patterns. Our work can be used to enrich the software security research field with security patch datasets that guide developers to better maintain software and improve software quality [13, 17].

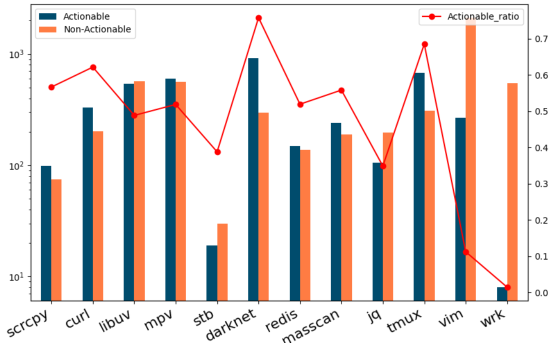

Figure 2: The number of warnings and fixed warning rate in the repositories.

5 LIMITATIONS

In this work, we begin by using Cppcheck to demonstrate the feasibility of constructing actionable warning datasets. Specifically, we have implemented only the incremental analysis algorithm of Cppcheck, and the dataset comprises solely C language repositories. However, our framework is designed for easy extensibility and seamless integration with other static analysis tools.

A key aspect of our approach consists of identifying the disappearance of warnings through a combination of static program analysis and incremental analysis of reports. However, a potential limitation of our approach lies in the assumption that the disappearance of warnings is solely due to their repair. To mitigate this concern, we implement a filtering mechanism that concentrates solely on the file where the warning was originally reported. This approach is intended to ensure that our analysis accurately reflects direct corrections to the warnings, thus avoiding misinterpretation due to irrelevant changes in other parts of the code base.

6 CONCLUSION AND FUTURE WORK

Identifying actionable warnings through machine learning has the potential to significantly enhance the utility of static analysis in the industry. However, there has been a notable lack of datasets to support this critical area of research. To address this need, we have presented AW4C, a large-scale, commit-aware dataset containing 38,134 real-world actionable warnings mined from over 500 popular open-source C repositories. We designed a novel mining framework to accurately map warnings to commits based on incremental analysis.

Our future endeavors will focus on expanding our tool's compatibility with other version control systems and integrating other static analysis tools into our framework. This extension aims to evolve our dataset into an even larger and more diversified resource that encompasses a variety of programming languages and tools, thereby significantly advancing research in software analysis and quality assurance.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (No. 62372071), the Fundamental Research Funds for the Central Universities (No. 2022CDJDX-005), the Chongqing Technology Innovation and Application Development Project (No. CSTB2022TIAD-STX0007 and No. CSTB2022TIAD-KPX0067) and the Natural Science Foundation of Chongqing (No. CSTB2023NSCQ-MSX0914).

References

[1] Enas A. Alikhashashneh, Rajeev R. Raje, and James H. Hill. 2018. Using Machine Learning Techniques to Classify and Predict Static Code Analysis Tool Warnings. In *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA).* https://doi.org/10.1109/aiccsa.2018.8612819

[2] Anon. [n. d.]. *Reducing False Positives of Static Program Analysis in Industry.* https://conf.researchr.org/getImage/ase-2023/orig/Challenge.pdf

[3] Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. 2022. Deep Learning based Vulnerability Detection: Are We There Yet? *IEEE Transactions on Software Engineering* (Sep 2022), 3280–3296. https://doi.org/10.1109/tse.2021.3087402

[4] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. 2014. Fine-grained and accurate source code differencing. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering.* https://doi.org/10.1145/2642937.2642982

[5] Xiuting Ge, Chunrong Fang, Jia Liu, Mingshuang Qing, Xuanye Li, and Zhihong Zhao. 2023. An unsupervised feature selection approach for actionable warning identification. *Expert Systems with Applications* 227 (Oct 2023), 120152. https://doi.org/10.1016/j.eswa.2023.120152

[6] Quinn Hanam, Lin Tan, Reid Holmes, and Patrick Lam. 2014. Finding patterns in static analysis alerts: improving actionable alert ranking. In *Proceedings of the 11th Working Conference on Mining Software Repositories.* https://doi.org/10.1145/2597073.2597100

[7] Sarah Heckman and Laurie Williams. 2011. A systematic literature review of actionable alert identification techniques for automated static code analysis. *Information and Software Technology* (Apr 2011), 363–387. https://doi.org/10.1016/j.infsof.2010.12.007

[8] HongJin Kang, KhaiLoong Aw, and David Lo. [n. d.]. Detecting False Alarms from Automatic Static Analysis Tools: How Far are We? ([n. d.]).

[9] Anant Kharkar, RoshanakZilouchian Moghaddam, Matthew Jin, Xiaoyu Liu, Xin Shi, Colin Clement, and Neel Sundaresan. [n. d.]. Learning to Reduce False Positives in Analytic Bug Detectors. ([n. d.]).

[10] Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Yawei Zhu, and Zhaoxuan Chen. 2022. SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities. *IEEE Transactions on Dependable and Secure Computing* (Jul 2022), 2244–2258. https://doi.org/10.1109/tdsc.2021.3051525

[11] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. 2018. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. In *Proceedings 2018 Network and Distributed System Security Symposium.* https://doi.org/10.14722/ndss.2018.23158

[12] Tukaram Muske and Alexander Serebrenik. 2021. Survey of Approaches for Postprocessing of Static Analysis Alarms. *ACM Computing Surveys,ACM Computing Surveys* (Oct 2021).

[13] Antonino Sabetta and Michele Bezzi. 2018. A Practical Approach to the Automatic Classification of Security-Relevant Commits. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME).* https://doi.org/10.1109/icsme.2018.00058

[14] Junjie Wang, Song Wang, and Qing Wang. 2018. Is there a "golden" feature set for static warning identification?: an experimental evaluation. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement.* https://doi.org/10.1145/3239235.3239523

[15] Xueqi Yang, Zhe Yu, Junjie Wang, and Tim Menzies. 2021. Understanding static code warnings: An incremental AI approach. *Expert Systems with Applications* 167 (Apr 2021), 114134. https://doi.org/10.1016/j.eswa.2020.114134

[16] Rahul Yedida, HongJin Kang, Huy Tu, Xueqi Yang, David Lo, and Tim Menzies. 2022. How to Find Actionable Static Analysis Warnings: A Case Study with FindBugs. (May 2022).

[17] Yaqin Zhou and Asankhaya Sharma. 2017. Automated identification of security issues from commit messages and bug reports. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering.* https://doi.org/10.1145/3106237.3117771

[18] Yaqin Zhou, Jing Kai Siow, Chenyu Wang, Shangqing Liu, and Yang Liu. 2022. SPI: Automated Identification of Security Patches via Commits. *ACM Transactions on Software Engineering and Methodology* (Jan 2022), 1–27. https://doi.org/10.1145/3468854