Towards Integration of Syntactic and Semantic Vulnerability Patterns

Lal Akhter, Muhammad Taimoor Khan, George Loukas and Georgia Sakellari
Centre for Sustainable Cyber Security
School of Computing and Mathematical Science
University of Greenwich, UK
Email: {l.m.akhter, m.khan, g.loukas, g.sakellari}greenwich.ac.uk

***Abstract*—This paper advances the field of software security by proposing an integrated approach for analysing both syntactic and semantic vulnerability patterns. Utilising a detailed vulnerability and attack library alongside a verification tool for language-neutral threat assessment, this study enhances the detection and mitigation of security threats in diverse programming environments. The research builds upon and refines previous work by employing Structured Threat Information eXpression (STIX) objects and XPath for syntactic analysis and introduces advanced semantic error detection techniques. A specialised tool developed and demonstrated previously to model vulnerability patterns from the MITRE database for comprehensive analysis to demonstrate the practical application of this research is now enhanced to add new features. This paper outlines the enhancements in the integrated analysis tool and shows its current features of detecting semantic vulnerability patterns using Infer. It also gives details of future development plans, which is the development of a web version, aiming to increase accessibility and utility. Highlighting the significance of a holistic vulnerability analysis approach, the research underscores the potential for future applications in securing open-source projects and broader software development practices.**

I. Introduction

The integration of technologies like 5G/6G, (I)IoT, and Edge computing into digital systems, involving diverse software across various programming languages [1], [2], significantly increases security challenges by expanding the attack surface with opaque components from different origins. Current research focuses on language-specific vulnerability analysis [3], [4], [5], [6], [7], often leading to false positives with machine learning techniques [8], [4], and existing verification methods are restricted to certain attacks and applications, lacking the capability to detect vulnerabilities across varied programming environments [9], [10]. This research presents a new approach to developing inherently secure applications through a language-neutral vulnerability analysis. It involves building a detailed library of vulnerabilities and attacks, their relationships, and a verification tool to assess application threats. This language-independent method is a first of its kind in utilising semantic models for cross-language vulnerability analysis, aims to identify threats in software across all mainstream programming languages. This research takes a different approach from tools like Boogie, which assists in creating verification tools. Instead, it focuses on a model-driven strategy by converting converting models into the STIX language. STIX objects are then used in the already available tools, leveraging the existing capabilities of PMD and Infer for its functionality. Our research introduced a methodology for vulnerability analysis, initially focusing on modelling syntactic vulnerability patterns. This approach utilised Structured Threat Information eXpression (STIX) [11] objects and XPath-based rule definitions to effectively identify and analyse vulnerabilities within Java code samples. Employing the PMD Source code analyser, our methodology demonstrated its capability to detect instances that either conform to or violate specific known vulnerability and attack models [12], thereby providing a foundational step towards understanding and mitigating security threats in software sys-

tems. Furthermore, we developed a prototype tool that identifies vulnerabilities and generates actionable threat information, facilitating the potential for automatic vulnerability recovery by systems like ARMET. While our initial efforts concentrated on syntactic patterns, we highlighted the ongoing extension of our methodology to encompass semantic patterns. This expansion aims to achieve a more comprehensive analysis of vulnerabilities, thereby enhancing the detection, understanding, and mitigation of security threats by detecting semantic error patterns.

II. Background

A. The Methodology Revisited

The methodology includes the following high-level workflow steps:

1. Modelling In this phase, we construct a model for syntactic and semantic vulnerabilities, framing them within a logical state relation. This model is abstract yet sufficiently detailed to encapsulate the necessary nuances of language semantics. Crucially, the models are amenable to software analysis for any target language. 2. Identification Rules From the given model, this step automatically generates the rules to identify the vulnerabilities. The syntactic rules describe the patterns based on the various constructs of the target language, e.g., specific API usage, method calls, object instantiation, and hard-coded parameters, to name a few. The semantic rules describe the patterns as a relationship between semantics of the target language constructs and the vulnerability semantics. 3. Analysis Based on the generated identification rules, this step automatically performs static analysis of the software given a specific target language to identify vulnerabilities. For syntactic rules, we extend PMD that employs syntactic pattern matching to detect syntactic patterns as given rules. For semantic rules, we extend Infer that employs verification to detect semantic patterns.

B. Syntactic vs Semantic Vulnerability Patterns

Syntactic vulnerability patterns are identified by matching specific syntax within the code to known vulnerability patterns. These patterns are systematically catalogued using the Structured Threat Information eXpression (STIX) format, subsequently translated into Xpath Query (XQ) expressions. The PMD tool is enhanced to execute these rules against code segments to determine the presence of these predefined patterns [13], [14]. Conversely, semantic error detection focuses on the correctness of the code's semantics. It aims to identify issues that arise from incorrect code logic, such as null pointer dereferences and leaks of resources or memory. These issues are particularly detrimental in mobile environments and are analysed using separation logic and bi-abduction techniques. This method delves deeper into the code's logic, beyond mere syntactic matching, to uncover vulnerabilities that could lead to significant operational problems [15].

C. Importance of Integrating Syntactic and Semantic Vulnerability Analysis

Integrating syntactic and semantic analysis is crucial for comprehensive code evaluation. This holistic approach ensures that code is rigorously tested against known vulnerability patterns and semantic correctness. By combining these analyses, developers can achieve a more robust detection of potential vulnerabilities, enhancing the security and reliability of the software [16], [17].

III. Case Studies and Tool Development

A. Application of Integrated Analysis in Real-World Scenarios

In practical application, our methodology has been applied to real-world scenarios, leading to the development of a specialised tool. This tool uses the MITRE database to extract vulnerability patterns, which are then modelled and translated into XPath queries. These queries are executed using the PMD tool, with the results fed back into our system for results display. Additionally, the tool can perform checks for semantic errors, offering a comprehensive analysis of potential vulnerabilities. Through this integrated approach, we aim to bridge the gap between theoretical vulnerability analysis and practical, actionable insights, potentially making a significant contribution to the field of software

security. However, it's important to note that this claim would benefit from further substantiation, such as through experimental evaluation by running the tool on opensource projects source code, which is part of our future research plan. [18]. It's also important to note that the current version represents an advancement from its predecessor, focusing primarily on foundational capabilities and effectiveness in smaller codebases. We recognize the critical nature of performance and scalability, especially for larger software projects, and acknowledge that these areas require further enhancement. Addressing these performance improvements and ensuring the tool's scalability for extensive software projects are integral parts of our future research agenda.

B. Enhancement to the Existing Tool

The tool, which is now named 'GRE Code Analyser, or GCA for short', is the under-development tool to practically implement the research. An earlier version of GCA was presented in a paper, and the subsequent presentation was at FAACS Istanbul in September 2023. This paper presents the most recent version of GCA, with all future references pertaining to this version of the tool.. The new version of GCA features a user-friendly main interface, which is an enhanced version of the previous interface, to streamline vulnerability analysis as shown in Figure 1. This interface is the gateway for users to interact with the tool's capabilities, facilitating a seamless experience from start to finish. Users now have the enhanced capability to choose an entire folder containing the source code for analysis, moving beyond the previous limitation of selecting only individual files. The capability of GCA to accept a complete folder containing code files as opposed to writing code within the interface introduces the ability to analyse a comprehensive source code folder, enabling the tool to process all contained files simultaneously. After choosing the desired folder, users can begin the analysis by concentrating on a single error type or expanding their scrutiny to include all error types listed in the tool's dropdown menu of available error options.

Once the analysis is complete, the tool compiles the findings into a text file. This file presents the results in a clear and concise manner, making it easy for users to review and interpret the detected vulnerabilities. This feature ensures that users have tangible output from their analysis efforts, which can then be used for further review or remediation activities.

In addition to syntactic error detection, the tool also offers the capability to check for semantic errors. Users can analyse either a single file or an entire folder for semantic vulnerabilities. This flexibility ensures that the tool can accommodate various user needs and scenarios. A dedicated folder selection dialogue enhances the user experience by allowing for easy navigation and selecting the target folder for semantic error analysis. This intuitive interface simplifies the process of specifying the scope of the analysis, ensuring that users can efficiently direct the tool's focus to the relevant codebase. Similar to the result for syntactic analysis, upon completing the semantic error analysis, the tool displays the results, providing users with detailed insights into the identified issues, see Figure 2. This feedback is crucial for understanding the code's semantic integrity and identifying potential areas of improvement or concern.

C. Tool Development for Integrated Analysis

The forthcoming version of our tool represents a significant leap forward in vulnerability analysis technology. It will feature an integrated framework capable of concurrently running syntactic and semantic error detection processes. This unified approach ensures a more comprehensive analysis, allowing users to efficiently identify a broader range of vulnerabilities within their source code. In addition to the integrated tool, we are also developing a web version. This online platform, which will be made publicly available, will make the tool accessible from anywhere, allowing users to conduct analyses without the need for local installations. The web version is designed to cater to the needs of a diverse user base, offering the same robust analysis capabilities in a more convenient format.

IV. Conclusion and Future Work

Reflecting on the screenshots above, the development and enhancement of GCA clearly illustrate

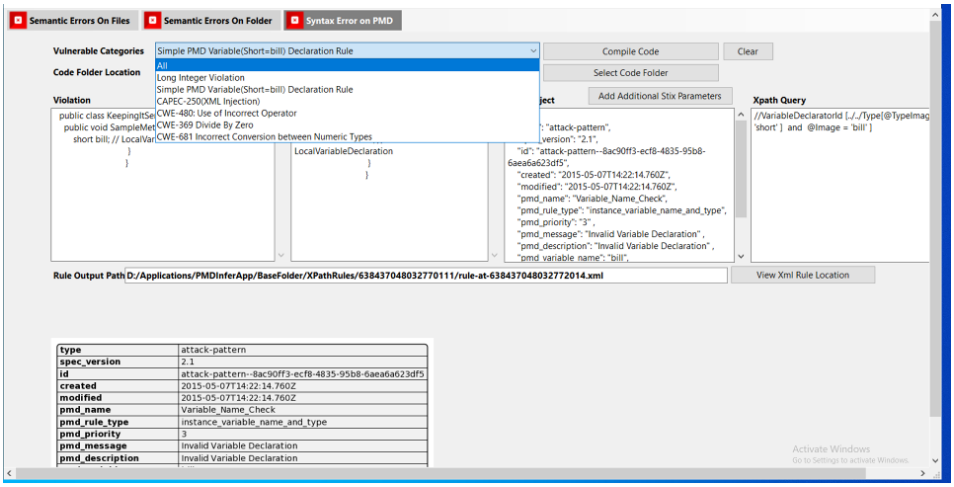

Fig. 1. Tool Interface

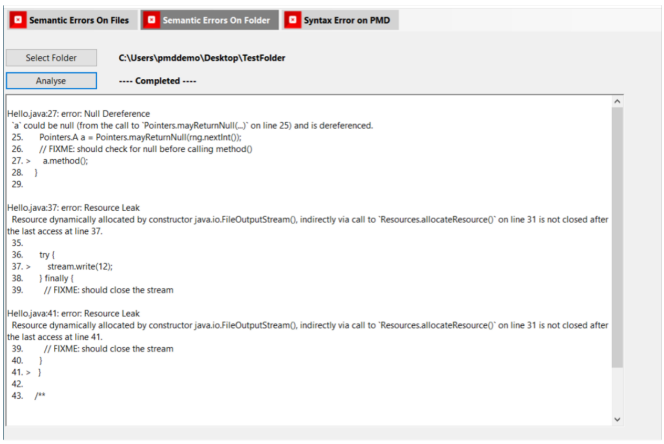

Fig. 2. Analysis Results

the significance of adopting a holistic approach to vulnerability analysis. By integrating syntactic and semantic error detection, we offer a more thorough method for identifying potential security risks. This integrated approach not only enhances the accuracy of vulnerability detection but also streamlines the analysis process, making it more efficient and user-friendly.

There is significant potential for applying our tool to a wide array of open-source projects. Our aim is to utilise the tool to scrutinise the source code of these projects, identifying vulnerabilities susceptible to known attacks and uncovering any semantic vulnerabilities. This application will demonstrate the tool's effectiveness in a real-world context and contribute to open-source software's security and integrity. Through this endeavour, we anticipate uncovering valuable insights that will drive further research and development in software security, paving the way for more secure software development practices.

REFERENCES

[1] J. Mota, M. Giunti, and A. Ravara, "Java typestate checker," in *International Conference on Coordination Languages and Models*. Springer, 2021, pp. 121–133.

[2] L. Wartschinski, Y. Noller, T. Vogel, T. Kehrer, and L. Grunske, "Vudenc: vulnerability detection with deep learning on a natural codebase for python," *Information and Software Technology*, vol. 144, p. 106809, 2022.

[3] N. Saccente, J. Dehlinger, L. Deng, S. Chakraborty, and Y. Xiong, "Project achilles: A prototype tool for static method-level vulnerability detection of java source code using a recurrent neural network," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*. IEEE, 2019, pp. 114–121.

[4] Y. Luo, W. Xu, and D. Xu, "Detecting integer overflow errors in java source code via machine learning," in *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2021, pp. 724–728.

[5] S. Chen, N. Jiang, Z. Wu, and Z. Wang, "Jsvulexplorer: a javascript vulnerability detection model based on transfer learning," in *Fifth International Conference on Computer Information Science and Artificial Intelligence (CISAI 2022)*, vol. 12566. SPIE, 2023, pp. 1071–1079.

[6] Y. Zhang and D. Liu, "Toward vulnerability detection for ethereum smart contracts using graph-matching network," *Future Internet*, vol. 14, no. 11, p. 326, 2022.

[7] P. Tolmach, Y. Li, S.-W. Lin, Y. Liu, and Z. Li, "A survey of smart contract formal specification and verification," *ACM Computing Surveys (CSUR)*, vol. 54, no. 7, pp. 1–38, 2021.

[8] G. Lin, W. Xiao, J. Zhang, and Y. Xiang, "Deep learning-based vulnerable function detection: A benchmark," in *Information and Communications Security: 21st International Conference, ICICS 2019, Beijing, China, December 15–17, 2019, Revised Selected Papers 21*. Springer, 2020, pp. 219–232.

[9] A. Nirumand, B. Zamani, B. Tork-Ladani, J. Klein, and T. F. Bissyandé, "A model-based framework for inter-app vulnerability analysis of android applications," *Software: Practice and Experience*, vol. 53, no. 4, pp. 895–936, 2023.

[10] S. Liu, G. Bai, J. Sun, and J. S. Dong, "Towards using concurrent java api correctly," in *2016 21st International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 2016, pp. 219–222.

[11] S. Barnum, "Standardizing cyber threat intelligence information with the structured threat information expression (stix)," *Mitre Corporation*, vol. 11, pp. 1–22, 2012.

[12] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, "Mitre att&ck: Design and philosophy," in *Technical report*. The MITRE Corporation, 2018.

[13] S. Wang, "Develop and evaluate a security analyzer for finding vulnerabilities in java programs."

[14] D. Al-Ashwal, E. Z. Al-Sewari, and A. A. Al-Shargabi, "A case tool for java programs logical errors detection: Static and dynamic testing," in *2018 International Arab Conference on Information Technology (ACIT)*. IEEE, 2018, pp. 1–6.

[15] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler, "A few billion lines of code later: using static analysis to find bugs in the real world," *Communications of the ACM*, vol. 53, no. 2, pp. 66–75, 2010.

[16] J. Berdine, C. Calcagno, and P. W. O'hearn, "Smallfoot: Modular automatic assertion checking with separation logic," in *Formal Methods for Components and Objects: 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1-4, 2005, Revised Lectures 4*. Springer, 2006, pp. 115–137.

[17] C. Calcagno, D. Distefano, P. O'Hearn, and H. Yang, "Compositional shape analysis by means of bi-abduction," in *Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2009, pp. 289–300.

[18] J. Constine, "Facebook acquires assets of uk mobile bug-checking software developer monoidics."