MJSAT

Malaysian Journal of Science and Advanced Technology

journal homepage: https://mjsat.com.my/

AI-Enhanced Static Code Analyzer for Secure Coding Support

Hor Jia Jie[1], Maisarah Mansor *[1,2], and Ranjit Singh Sarban Singh[3,4]

[1]*School of Computing and Artificial Intelligence, Faculty of Engineering and Technology, Sunway University, No. 5, Jalan Universiti, Bandar Sunway, 47500 Selangor Darul Ehsan, Malaysia.*

[2]*Research Centre for Nanomaterials and Energy Technology (RCNMET), Faculty of Engineering and Technology, Sunway University, No. 5, Jalan Universiti, Bandar Sunway, 47500 Selangor Darul Ehsan, Malaysia.*

[3]*School of Engineering, Faculty of Engineering and Technology, Sunway University, No. 5, Jalan Universiti, Bandar Sunway, 47500 Selangor Darul Ehsan, Malaysia.*

[4]*Research Centre for Humac-Machine Collaboration (HUMAC), Faculty of Engineering and Technology, Sunway University, No. 5, Jalan Universiti, Bandar Sunway, 47500 Selangor Darul Ehsan, Malaysia.*

KEYWORDS

Static Code Analysis
Artificial Intelligence
Secure Coding Education
Large Language Models
Explainable AI

ARTICLE HISTORY

Received 26 February 2026
Received in revised form 31 July 2026
Accepted 14 August 2026
Available online 28 August 2026

ABSTRACT

Many industrial security scanners including the popular Bandit tool discover security flaws; however, the information they give to the user is frequently technical and difficult to understand for those who are new to security. The goal of this project is to improve educational tools for secure programming that novice programmers can use by making static code checks easier to comprehend. A browser-based software tool called AegisCode was created using advanced mathematical techniques for counting and combinations. This tool makes use of Google's Gemini Large Language Model. While the current version utilizes a client-side simulation to provide immediate feedback, the proposed architecture details a Python Flask backend designed to run actual Bandit rule-based scanning to replace non-deterministic AI interpretations. The testing of the proof of concept showed it was performing well, and the whole analysis process was normally finished within five seconds. This process was also very accurate in producing well formatted JSON output which can be used by the user interface. This paper is innovative in its two-process theory, the conceptual underpinning for the translation of the technical diagnostics into the rich narrative stories for an education audience. Junior developers are provided with key security advice by the system so they can avoid security pitfalls. Learning how to programme using a tool such as this teaches developers good coding practices and appropriate methods of coding. It aids in promoting acceptable code practices. Future work will prioritize the transition to a deterministic backend and the integration of the Semgrep engine to expand language support beyond Python to include C++ and Java, ensuring compliance with OWASP and CWE standards.

1. INTRODUCTION

Software permeates every industry today and the integrity and security of such systems are vital to public safety and user trust. Unfortunately, many of the very severe vulnerabilities have their root in careless coding practices and a lack of developer knowledge about dangerous patterns. History has shown that the inability to patch simple vulnerabilities like unvalidated inputs or hard-coded credentials can have disastrous consequences, such as the 2017 Equifax breach, where 147 million people's data was exposed [1]. In view of this threat, the shift-left concept in software engineering promotes the integration of security checks at the earliest

*Corresponding author:
E-mail address: Maisarah Mansor < maisarahma@sunway.edu.my >.
https://doi.org/10.56532/mjsat.v6i3.729

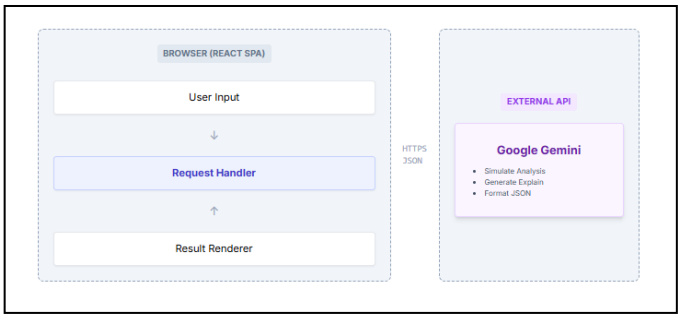
stages of the development life cycle [2]. Tools such as Bandit identify such patterns through source code analysis, but these are designed for professional use and most of their alerts are unintelligible to novices [3], [4]. Students often focus on functional output while being taught programming rather than secure coding principles. This inevitably leads to insecure habits being formed that harden quickly into long-term professional practice [5]. The shift-left concept suggests moving software security measures to an earlier phase in the software development lifecycle. This idea of moving to emphasize building cybersecurity capacity in Malaysia's nascent cybersecurity sector, spearheaded by institutions like Sunway University's School of Computing and Artificial Intelligence, foresees a path that would allow AegisCode to find its way into Malaysian university curricula and environments where this cybersecurity expertise is limited. Integration into the educational system would be a benefit that future graduates and even industry professionals could enjoy [6].

However, statistics demonstrate that nearly three out of four companies have experienced a data breach caused by previously known, unaddressed security flaws [7]. Even if audit tools' results can be complex for a beginner to grasp, an understanding of their importance is vital to any future coder's education. Now with the advent of Large Language Models (LLMs), it is possible to explain complicated information in a technical field. The AegisCode project addresses security skill gaps through use of the integration of static code analysis with an artificial intelligence narrative engine. This will provide the user with feedback in the range of two to five seconds; this is particularly important for student reflection as it enables the pupil to assess and adapt their own coding skills in real-time as they are working with the logic of the code. This is achieved using simulated security attacks and associated feedback which explains how and why security vulnerabilities occur and how to avoid them. Feedback is given to the user in two to five seconds. The advice is aimed at helping budding coders to better understand how to program computers securely, and to develop the skills to create software that is resilient to hacking attacks. The system educates by providing examples of bad code which children can learn from, as well as good code examples for students to follow. Developers of the future are more likely to use better quality security tools if the static code analysis process can be enhanced with the use of Artificial Intelligence (AI).

2. METHODS

This project was based on a Design Science Research (DSR) methodology which was undertaken in an iterative and incremental way. This means that the system would be developed in an iterative manner through prototyping, testing and refinement to enhance system reliability and clarity of the pedagogical interface. The implementation was based on the use of iterative and incremental methodology which is used to continuously prototype, test and refine to enhance the system reliability and the clarity of the interface [8]. methodological approach was mixed to differentiate the design and the system. The system should have a React front-end and a Python Flask back-end with Bandit. The Google Gemini API was selected for the prototype because the model is state-of-the-art in creating the contextualised, pedagogical stories needed to support novices' understanding of complex security topics [9]. This workflow and the relationship between these components are illustrated in the architectural diagram in Figure 1.

Fig. 1. Hybrid Client-Side Architecture



The AI should behave similar to the Bandit's rule-based engine, which searches for vulnerable code patterns such as hardcoded passwords, untrusted functions and lack of input validation. The main goal of this plan is to ensure that role-based behaviour and JSON schema are preserved. It is achieved by Prompt Engineering and fine-tuning. There are two roles that AI plays in the system: one is to generate diagnostic data, and the other is to teach security through useful, informative and contextual stories. A Security Scoring Mechanism was developed to secure code. This mechanism is based on a 100-point scale from which points are deducted in a very specific manner, according to the risk levels that OWASP Top 10 and CWE [10]. This makes the scoring not just heuristic, but it also draws on industry standards for risk assessment.

The approach was designed to test Python scripts and API errors. The tool uses a no-execution policy to guard against Remote Code Execution (RCE) and prompt injection attacks. Through the sole treatment of input code as text, the tool can avoid the execution of potentially harmful code, while being able to perform in-depth analysis of the code's structure [11]. During the System Performance Evaluation, the responsiveness of the API and the return of the JSON is considered. In the Usability Evaluation, the explanations of the pedagogical content are evaluated.

3. FINDINGS AND ANALYSIS

3.1 System Performance and Output Consistency

The analysis of the AegisCode system indicates that the system is stable, communicative, and relies on dependable components for its novice users. Testing the performance of the client-side components on typical student computers indicates that the client can very efficiently handle medium-sized Python scripts. The total time for the analysis is between two and five seconds. This relatively short time for processing is very important for the educational setting, as it allows the system to provide immediate feedback to the students.

The uniformity of the AI-generated JSON output was a big find when it came to system dependability because the frontend interface needs properly structured data to show diagnostic results. If the Gemini API sends back data that isn't properly formatted, the system could break down, and this is a known risk that comes with using foundation models for structured technical work. Following these iterative

improvements to fast engineering and schema compliance, the number of incorrectly formatted replies went down a lot. The high success rate with parsable JSON ensures that this tool can be used in typical educational settings and fulfills the need for such tools. Table 1 shows how well the system works and the safety of the data.

Table 1. System Performance and Data Consistency

Metric	Observation
Processing Time	2 to 5 seconds per analysis
JSON Success Rate	High success following prompt refinement
Interface Responsiveness	Stable with no visual jitter or rendering delays
Memory Usage	Minimal impact on browser performance

3.2 Dual-Output Interpretability and Usability

The second set of results further makes clear that the dual output of this system can help to reduce the interpretability gap, or the security knowledge gap, which makes it difficult for novices to understand the technical feedback that is often provided to them [12]. The feedback includes rule IDs, severity levels, and line references that will help these students to become more familiar with the type of information that is commonly provided in CI/CD pipelines. The AI also provides descriptions of the security ideas and the hazards that are presented in the technical output, giving the feedback to students the necessary instructional context that most traditional approaches do not provide.

Additionally, the results from the usability tests help to show that providing unprocessed technical output and instructional support leads to a reduction in cognitive overload [13], [14]. For instance, students can easily and effectively find the most significant security issues and the steps that will fix them if the output includes visual cues like color-coded severity indicators, such as red for high-risk vulnerabilities as shown in Figure 2. Not only does the technology help to uncover errors in the code, but it also provides workers with hints that can help them to improve their coding skills.

Fig. 2. Dual-Layer Output Model

4. DISCUSSION AND RECOMMENDATIONS

The development of AegisCode points to a critical success in closing the security knowledge gap through its transformation of technical diagnosis into pedagogical narratives. Table 2 shows the comparison between AegisCode and established developer tools. While traditional tools prioritize technical telemetry, AegisCode's novelty is rooted in dual-process theory, which bridges the interpretability gap by transforming raw technical telemetry into Explainable AI (XAI) narratives specifically designed for learners.

Table 2. Comparative Analysis of AegisCode vs. Existing AI-Assisted Developer Tools

Tool Name	Dual-Output Pedagogy	JSON Schema Enforcement	Security Scoring	Narrative Design	Ref
AegisCode	Primary (Technical & Narrative)	Strict via Iterative Prompting	100-Point Deduction System	Context-Rich Pedagogical Stories	This work
GitHub Copilot (Security Prompts)	Limited (Suggestion-based)	Not Specified	Standard Prompts	General Advice	[15]
SonarQube (Educational Plugins)	Limited (Plugin-dependent)	Standard Technical Format	Metric-based (Quality)	Technical Documentation	[16]
Semgrep (Educational Integrations)	Secondary (Remediation-focused)	Rule-based Output	Rule Severity	Technical Remediation	[17]

By building on the LLMs capability to explain technical topics, the system can provide immediate feedback to beginners [18], [19]. Despite the numerous benefits that this system provides, a few problems arise, primarily in the context of the fidelity of the present analysis engine. The most salient of these limitations is that there is no actual execution of Bandit in the system.

The outcome of these approaches is a reliance on non-deterministic interpretation that may not provide the same level of accuracy as the static analysis rules in existing code analysis tools [20]. In recognizing these technical risks, various risk mitigation techniques and strategies are employed. At first strategy, in recognizing technical risks such as AI hallucination, the project employs iterative prompt refining. This ensures the LLM maintains strict role-based behavior and adheres to the required JSON schema, thereby reducing the incidence of non-deterministic errors [21]. The second strategy is implementing Input Validation & Error Handling to ensure that the Python scripts have the correct format and that API requests do not fail. The third strategy is to implement a RCE mitigation technique that simply does not allow any execution of the submitted code. By treating the user's input as static text only, this approach prevents RCE and prompt injection. The fourth and final strategy is to use iterative prompt refining to ensure that the models comply with the given JSON schema and do not behave outside of the roles that they are given. The system only supports Python-specific prompts and weights. Current limitations include a dependency on active network stability and external API keys for client-side functionality. Future iterations will address this by implementing response caching and local analysis capabilities to enhance tool autonomy. While currently restricted to Python, a specific development timeline has been established to integrate the multi-language Semgrep engine, which will broaden the tool's utility across diverse academic curricula. Table 3 summarizes current limitations and provides proposed solutions.

Table 3. Summary of Current Limitations and Proposed Solutions

Identified Problem	Impact on User Experience	Recommended Future Action
Simulated Detection	Lacks deterministic rule fidelity	Integrate Flask backend for real Bandit execution
Language Restriction	Limited to Python scripts	Expand support to Java and C++ using Semgrep
API Dependency	Requires active network and keys	Implement response caching and local analysis
Basic Scoring	Lack of rigorous industry alignment	Formalize weights using OWASP Top 10 and CWE metrics to enhance XAI justification
Lack of Empirical Validation	Uncertainty regarding actual learning gains and cognitive impact	Implement a DSR plan featuring controlled user studies and pre/post-test evaluations

To transform AegisCode into a production-ready assistant, several recommendations are put forward for future work. First is the full implementation of backend architecture using Flask, which will enable the system to perform an actual Bandit scan and yield deterministic JSON results for AI interpretation. This backend will support on future pathways including University LMS platforms, IDE plugins (VS Code) and CI/CD pipelines to support the shift-left concept, exploring an institutional license and an open-source freemium model to ensure viability for academic institutions and SMEs and including response caching and local analysis to reduce dependency on network stability and external API keys. Second is the expansion of language scope; this would require the addition of multi-language engines such as Semgrep to support a broader swath of academic curricula. Students would be able to understand industry-standard criteria better if the security health score was in line with well-known standards like the OWASP Top 10 or CWE. The security scoring mechanism has been formalized to move beyond a purely heuristic approach. While the 100 point pedagogical starting point is maintained to aid in cognitive map construction, these deduction weights have been mapped to ensure alignment with industry-standard vulnerability assessments, providing a rigorous basis for the Security Health Score and transforms the scoring from a simple deduction system into an Explainable AI tool that reduces cognitive barriers by explaining the why behind each score reduction.

Implement a comprehensive DSR plan to bridge the interpretability gap between technical alerts and learner comprehension. This action incorporates controlled user studies using pre-test and post-test evaluations to quantify measurable learning gains, utilizes user comprehension surveys to assess reduced cognitive load and the facilitation of cognitive map construction, and performs experimental comparisons between raw Bandit output and AegisCode-assisted output to evaluate long-term secure coding retention among novice developers [22]. In the end, adding interactive features like static code previews, guided tutorials and progress tracking would greatly improve the tool's ability to teach. With these changes, the system will go from a proof-of-concept prototype to a full-fledged, scalable learning platform.

5. CONCLUSION

These findings suggest that a two-layered output is one possible solution to fill in the gap between complex technical and simple understandable alerts. It demonstrates that automated vulnerability scanning can be turned into a teaching session by providing raw vulnerability scan data with narratives. The idea proposed has a major limiting factor that it is based on non-deterministic AI simulation and is nowhere near as accurate as a deterministic scanning engine like Bandit because it is not rule-based. The second problem with the current prototype is that it is only applicable to Python and the API client is dependent upon the stability of the API. This project has shown that XAI can be used to overcome cognitive challenges for new programmers. The internalisation of OWASP and CWE classes in the formalised scoring system and double-layered output of AegisCode transforms a proof-of-concept to a scalable solution to support national cybersecurity capacity building. Future work will focus on adding a Flask-based backend to run Bandit, extending support to other languages such as Java and C++ through integration with Semgrep and making the tool a full-fledged learning platform for the next generation of secure coders. Finally, this project shows that AI-assisted static analysis can be implemented and is helpful for building national cybersecurity capacity in the next generation of Malaysian software developers.

REFERENCES

[1] I. Kabanov and S. E. Madnick, “A Systematic Study of the Control Failures in the Equifax Cybersecurity Incident,” SSRN Electronic Journal, 2020, doi: https://doi.org/10.2139/ssrn.3957272.

[2] B. K. Kaithe, “Shift Left Security: A Paradigm Shift in Software Development Security Integration,” Eur. J. Comput. Sci. Inf. Technol., vol. 13, no. 24, pp. 96–102, Apr. 2025. doi: https://doi.org/10.37745/ejcsit.2013/vol13n2496102.

[3] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H. R. Lipford, “Questions developers ask while diagnosing potential security vulnerabilities with static analysis,” in Proc. 10th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Foundat. Softw. Eng. (ESEC/FSE), 2015, pp. 248–259. doi: https://doi.org/10.1145/2786805.2786812.

[4] F. Schuckert, B. Katt, and H. Langweg, “Difficult SQLi Code Patterns for Static Code Analysis Tools.”

[5] A. Nance, K. Hay, and B. Bishop, “Secure Coding Education: Are We Making Progress?”

[6] “Malaysia - Cyber Security Strategy (2020-2024),” 2020. url: https://regulations.ai/regulations/RAI-MY-NA-MCSS2XX-2020.

[7] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck, “Modeling and discovering vulnerabilities with code property graphs,” in Proc. IEEE Symp. Security and Privacy, 2014, pp. 590–604. doi: https://doi.org/10.1109/SP.2014.44.

[8] A. Shrestha, A. Cater-Steel, M. Toleman, and T. Rout, “The role of international standards to corroborate artefact development and evaluation: Experiences from a design science research project in process assessment,” Commun. Comput. Inf. Sci., pp. 438–451, 2017. doi: https://doi.org/10.1007/978-3-319-67383-7_32.

[9] V. Aware, A. Pimparkar, C. Unavane, A. Wagh, P. Pandit, and A. Yenkikar, “AI Agent for Scientific Paper Analysis,” in Proc. IEEE Int. Conf. Intelligent Signal Process. Effective Commun. Technol.

(INSPECT), 2025, pp. 1–7. doi: https://doi.org/10.1109/INSPECT67393.2025.11350774.

[10] "OWASP Top Ten Web Application Security Risks." url: https://owasp.org/www-project-top-ten/.

[11] P. Kishore and M. B. M, "Evolution of client-side rendering over server-side rendering." Recent Trends in Information Technology and its Application, vol. 3 Issue 2. doi: https://doi.org/10.5281/zenodo.3930215.

[12] B. London and T. Joachims, "Control variate diagnostics for detecting problems in logged bandit feedback," 2022.

[13] W. E. Roberts, "The use of cues in multimedia instructions in technology as a way to reduce cognitive load," *Journal of educational multimedia and hypermedia*, vol. 26, no. 4, pp. 373–412, Oct. 2017.

[14] S. Kalyuga, P. Chandler, and J. Sweller, "When redundant on-screen text in multimedia technical instruction can interfere with learning," Hum. Factors, vol. 46, no. 3, pp. 567–581, 2004. doi: https://doi.org/10.1518/hfes.46.3.567.50405.

[15] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions," Dec. 2021. doi: https://doi.org/10.48550/arXiv.2108.09293.

[16] J. Escribano-Barreno, J. García-Muñoz, and M. García-Valls, "Integrated metrics handling in open source software quality management platforms," in Advances in Intelligent Systems and Computing, pp. 509–518, 2016. doi: https://doi.org/10.1007/978-3-319-32467-8_45.

[17] C. Lohest and A. Legay, "Improving security analysis rule set by relationship identification," in Proc. IEEE Int. Conf. Softw. Test. Verif. Valid. Workshops (ICSTW), pp. 297–300, 2024. doi: https://doi.org/10.1109/ICSTW60967.2024.00059.

[18] L. M. Cruz, G. Castelblanco, and P. D. Antonenko, "LLM-based System for Technical Writing Real-time Review in Urban Construction and Technology," *EPiC series in built environment*, May 2024, doi: https://doi.org/10.29007/d9j3.

[19] S. Riazi and P. Rooshenas, "LLM-driven feedback for enhancing conceptual design learning in database systems courses," in Proc. 56th ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE TS), pp. 1001–1007, Feb. 2025. doi: https://doi.org/10.1145/3641554.3701940.

[20] M. Miao, A. Mordahl, D. Soles, A. Beideck, and S. Wei, "An extensive empirical study of nondeterministic behavior in static analysis tools," in Proc. Int. Conf. Softw. Eng. (ICSE), pp. 1064–1076, 2025. doi: https://doi.org/10.1109/ICSE55347.2025.00125.

[21] S. Abdelnabi, K. Greshake, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in Proc. 16th ACM Workshop Artif. Intell. Secur. (AISec), pp. 79–90, Nov. 2023. doi: https://doi.org/10.1145/3605764.3623985.

[22] L. Cohen, L. Manion, and K. Morrison, Research Methods in Education. Routledge, 2013.