International Journal of Scientific Research & Engineering Trends
Volume 10, Issue 5, Sep-oct-2024, ISSN (Online): 2395-566X

Securing AI-Assisted Cloud Engineering: Guardrails for Copilot-Generated Iac and CI/CD Changes to Prevent Vulnerability Injection

Sunil Anasuri[1], Komal Manohar Tekale[2]
[1]Sr. Specialist SAP OTC MicroVention Inc. USA
[2]Sr Dev Ops Engineer Auto club of Southern California USA

Abstract- The quick pace of AI coding assistant adoption in cloud engineering has greatly led to the creation of Infrastructure-as-Code (IaC) and CI/CD pipelines. Nevertheless, AI-generated setting may readily imply security misconfigurations, insecure defaults and violations of the policy that can be transmitted straight into production cloud environments. Such risks are especially acute in those organizations that deal with regulated and high-assurance industries, whose misconfigured resources can cause data breaches, privilege increases, and violation of the rules. Conventional security review procedures are too sluggish and manual to follow through with the AI-assisted development processes, which resulted in a pressing need of automated preventive security mechanisms. The paper presents a recommendation in the form of the AI Guardrailed Cloud Engineering Framework (AGCEF) that is a proactive security model that involves the imposition of guardrails on AI-generated IaC and CI/CD artifacts prior to the deployment. AGCEF combines policy-as-code checking, matching of vulnerability signatures, semantic intent checking with LLM and a quantitative risk scoring system, which identifies and thwart insecure configurations at design time. Through experimental analysis, it is shown that AGCEF is significantly better in comparison to current AI-based methods of vulnerability detection because it offers higher vulnerability prevention, lowers false negatives, less manual review, and enhances the safety of deployment. The framework allows organizations to use AI copilots to enhance productivity and maintain high levels of cloud security and compliance, hence restoring the balance between the speed of AI-assisted development and AI-assisted operations in the cloud.

Keywords – AI-assisted cloud engineering, Infrastructure as Code security, CI/CD pipeline security, AI code guardrails, DevSecOps automation, policy-as-code, cloud misconfiguration prevention, LLM security validation, secure AI-generated code, vulnerability injection prevention.

I. INTRODUCTION

Cloud computing has revolutionized the way contemporary organizations design, implement and scale out digital services [1]. As the use of Infrastructure-as-Code and automated CI/CD pipelines has taken centre stage, manual configuration of cloud engineering has been replaced by software-defined infrastructure [2]. This development has made it much easier to achieve agility and repeatability, but it has also made it much more challenging to handle secure settings in dynamic and distributed environments [3]. In the recent past, AI coding assistants have been incorporated in cloud engineering processes to speed up the process of developing deployment templates, configuration files, and pipeline scripts. These AI copilots are able to produce Terraform modules, Kube manifests, and YAML-based CI/CD definitions in seconds, saving much time on developing it [4]. Nevertheless, the high pace of infrastructure code production without any human verification creates a new line of risks that the conventional DevOps procedures were not capable of managing [5].

In contrast to the traditional software bugs, cloud misconfigurations can present the key assets of storage buckets, virtual machines, identity roles, and network facilities directly. Outputs of AI can contain highly permissive access controls, turned-off encryptions, discovered secrets, or unsafe container settings [6]. Vulnerabilities can be delivered to production environments at a machine pace when automatic pushes of such artifacts are taken through CI/CD pipelines. The fundamental issue is caused by the fact that AI-supported development speed is inconsistent with the manual and slower security reviewing process [7]. It is not feasible that human reviewers will have to review all AI generated configuration changes, particularly in large organizations with hundreds of deployments per day [8].

© 2024 IJSRET

This weakness forms a strong necessity of automated security guardrails that can simultaneously function as fast as AI generation [9].

The current security tools are more or less at the point of detecting vulnerabilities upon deployment, or later in the pipeline [10]. Although static analysis and vulnerability scanners are useful, they frequently do not have much contextual knowledge of developer intent and are not closely connected with AI generation processes. Consequently, they can either overlook minor misconfigurations or produce too many false positives which flood the engineers [11].

Moreover, AI-generated configurations are also a special case as they can be syntactically correct and logically organized and yet break security best practices [12]. Even conventional rule-based scanners are not always able to understand whether the created infrastructure is in line with the targeted security policies. It points out the necessity of smart validation mechanisms that are able to comprehend technical configuration as well as semantic intent [13].

To overcome the problems, this paper proposes the AI Guardrailed Cloud Engineering Framework (AGCEF), a preventive model of security architecture that considers AI-supported workflows in the cloud. AGCEF integrates automated guardrails within the development pipeline that helps to assess the AI-generated IaC and CI/CD artifacts and only allow them to be deployed upon acceptance [14]. The framework integrates several levels of analysis, such as policy-as-code compatibility checks, familiar vulnerability pattern matching, and code behavior-intent comparison under semantic verification that is based on LLM [15]. These elements are used to provide a quantitative risk score to every artifact, and decide to approve, review, or block the artifact altogether [16].

AGCEF aligns with the concepts of DevSecOps and shift-left security by moving security validation to day one of the AI-assisted development of an application program [17]. The framework does not only apply the reactive approach to detecting vulnerability, but it rather concentrates on eliminating the vulnerability at the point of injection [18]. The real-world applicability of this proactive strategy involves the design of a guardrailed AI-assisted cloud engineering system, a decision model that is based on risks and is used to control the deployment gating, and a thorough assessment of demonstrating a stronger vulnerability prevention regarding the AI-based detection methods that are currently deployed. Collectively, these contributions offer a scalable, and feasible way forward towards organizations with the intention to integrate AI coding assistants into cloud engineering processes safely.

II. LITERATURE REVIEW

N. Thalji et al. [1] introduced AE-Net, an autoencoder-based deep learning framework designed for SQL injection attack detection. The model extracts deep latent features from web traffic data using autoencoders and improves classification accuracy compared to traditional signature-based methods. AE-Net demonstrates enhanced detection capability for both known and unknown attack patterns by leveraging unsupervised feature learning. However, its performance may depend heavily on training data diversity. H. Wang et al. [2] proposed SoFI, a security property-driven framework for vulnerability assessment of integrated circuits against fault-injection attacks.

The model analyzes hardware-level security properties and simulates potential fault scenarios to detect vulnerabilities early in the design phase. This approach enhances hardware trustworthiness but requires detailed domain-specific knowledge and computational resources for analysis.

K. K. Singamaneni et al. [3] presented an improved QCP-ABE (Ciphertext-Policy Attribute-Based Encryption) framework enhanced with dynamic polynomial integrity verification for cloud data security. The approach aims to strengthen access control and ensure data integrity in large-scale cloud environments. While effective for secure data sharing, the framework introduces additional computational overhead due to complex encryption operations. D. V. Medhane et al.

[4] proposed a blockchain-enabled distributed security framework integrating edge cloud computing and software-defined networking (SDN) for IoT environments. The model leverages blockchain for decentralized trust and secure data exchange while enabling scalable network management. Despite improved security, blockchain integration may increase latency and resource consumption.

S. Das et al. [5] introduced a multi-authority CP-ABE-based access control model designed for IoT-enabled healthcare systems. The framework provides fine-grained access control by distributing authority among multiple entities, enhancing privacy and data confidentiality. However, key management complexity and computational cost may limit real-time deployment. F.

Rustam et al. [6] proposed a deep ensemble-based framework for network intrusion detection. The model combines multiple deep learning algorithms to improve classification accuracy and reduce false positives. Ensemble learning enhances robustness but increases training complexity and computational requirements.

R. Madhvan et al. [7] provided a comprehensive overview of malware injection attacks, including techniques, impacts, and

© 2024 IJSRET

countermeasures. The study analyzes attack vectors and existing defense mechanisms, offering insights into emerging threats. However, it is primarily a survey and does not propose a specific implementation model. T. Sheth et al. [8] proposed a SQL injection detection system using Q-learning agents, applying reinforcement learning to dynamically identify malicious patterns. The system prioritizes adaptive learning and improves detection over time. Limitations include training complexity and dependence on reward function design. The limitations of the traditional models are shown in Table 1.

Table 1: Limitations of Traditional Models

Author Name & Ref No.	Proposed Model	Algorithm Used	Limitations
N. Thalji et al. [1]	AE-Net (Autoencoder-based SQL Injection Detection)	Deep Autoencoder, Deep Learning Classification	Requires large training datasets; computationally intensive
H. Wang et al. [2]	SoFI Vulnerability Assessment Framework	Security property analysis, fault injection simulation	Hardware-specific complexity; high analysis overhead
K. K. Singamaneni & S. N. Pasala [3]	Dynamic Polynomial Integrity QCP-ABE Framework	Attribute-Based Encryption (ABE), Polynomial Integrity	Increased encryption/decryption latency
D. V. Medhane et al. [4]	Blockchain-enabled IoT Security Framework	Blockchain + SDN + Edge Computing	Scalability and latency challenges
S. Das & S. Namasudra [5]	Multi-authority CP-ABE Access Control	Ciphertext Policy ABE	Key management complexity; performance overhead
F. Rustam et al. [6]	Deep Ensemble Network Attack Detection	Ensemble Deep Learning Models	High computational cost; complex training
R. Madhvan & M. F. Zolkipli [7]	Malware Injection Attack Survey	Analytical review (no specific algorithm)	No implementation or experimental validation
T. Sheth et al. [8]	Q-learning based SQL Injection Detection	Reinforcement Learning (Q-learning)	Requires careful reward design; slow initial learning

III. PROPOSED MODEL

Infrastructure-as-Code (IaC), configuration scripts and CI/CD pipeline workflows AI-based coding assistants, including Copilot-like tools, are increasingly being used by cloud engineers to generate this type of code. As much as these tools can expedite the development process and save on manual effort, they also present a novel category of security risks: artificial misconfigurations, insecure defaults, and policy breaches can be deployed to production systems without even being intentionally introduced to production systems. When errors occur in large-scale cloud systems - particularly those handling workloads of U.S. federal, healthcare, finance, and critical infrastructure - they can result in exposed storage buckets, excessively overly permissive IAM roles, no encryption or unpatched container images. The conventional security reviews would be unable to match the AI-assisted development pace, involving automated and proactive guardrails.

We are going to solve this problem by introducing an AI Guardrailed Cloud Engineering Framework (AGCEF) that applies security and compliance controls to AI-generated IaC and CI/CD artifacts prior to their deploymentThe framework is based on policy-as-code validation, code analysis, semantic code analysis, intent verification with the help of AI, and a risk-scoring system to eliminate vulnerabilities in a system introduced at design time prior to occurrence.By integrating such controls right into development pipelines, AGCEF guarantees that none of the productivity advances of AI copilots are obtained at the expense of a compromised cloud security.

The general process of securing Generative AI model is shown in Figure 1.

© 2024 IJSRET

International Journal of Scientific Research & Engineering Trends
Volume 10, Issue 5, Sep-oct-2024, ISSN (Online): 2395-566X

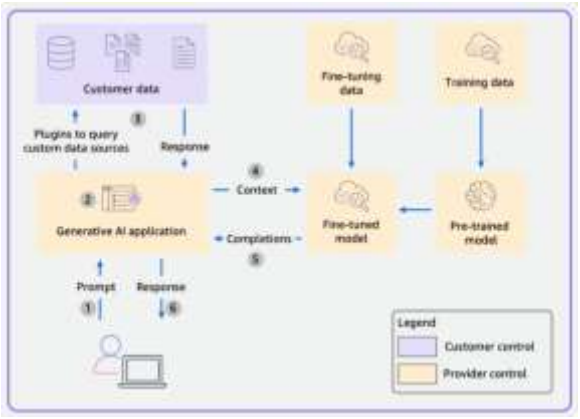

Fig 1: General Process of Securing Generative AI

Algorithm: Guardrailed AI IaC Verification Protocol (GAIIVP)

Inputs

- G: AI-generated code artifacts (IaC scripts, YAML pipelines)
- P: Security and compliance policies (IAM least privilege, encryption rules)
- V: Known vulnerability database (CVEs, misconfiguration patterns)
- M: ML/LLM verification models
- H: Historical secure configuration patterns

Outputs

- Verified code G^*
- Risk score R_g
- Deployment decision D_g

Step 1: Syntax & Structure Validation
Parse AI-generated artifact:

$$S_g = \text{Parse}(G)$$

If syntax invalid → reject.
Step 2: Policy-as-Code Compliance Check
For each policy $p_i \in P$:

$$C_i = \begin{cases} 1 & \text{if } G \vDash p_i \\ 0 & \text{otherwise} \end{cases}$$

Step 3: Vulnerability Pattern Matching
Scan configurations against known vulnerability signatures:

$$V_g = \sum_k \mathbb{1}(G \text{ matches } v_k)$$

Step 4: LLM-Based Semantic Intent Verification
Use LLM to interpret developer intent and compare with code behavior:

$$I_g = M_{\text{intent}}(G, \text{context})$$

Mismatch score:

$$\Delta_g = \| I_g - S_g \|$$

Step 5: Risk Scoring
Compute composite risk score:

$$R_g = \alpha(1 - \bar{C}) + \beta V_g + \gamma \Delta_g$$

Where:

- $\bar{C} = \frac{1}{|P|}\sum C_i$

Step 6: Decision Gate

$$D_g = \begin{cases} \text{Approve} & R_g < \tau_1 \\ \text{Require Review} & \tau_1 \le R_g < \tau_2 \\ \text{Block} & R_g \ge \tau_2 \end{cases}$$

Step 7: Continuous Learning Feedback
After deployment:

$$\theta_{new} = \theta_{old} - \eta \nabla \mathcal{L}(\text{PostDeployIssues}, R_g)$$

Mathematical Models

- **Policy Compliance Ratio**
- $\bar{C} = \frac{1}{n}\sum_{i=1}^{n} C_i$

Vulnerability Match Indicator

- $V_g = \sum_{k=1}^{m} \mathbb{1}(G \supseteq v_k)$

Semantic Mismatch Score

- $\Delta_g = \| \text{Embedding}_{intent} - \text{Embedding}_{code} \|_2$

- **Overall Risk Function**
- $R_g = \alpha(1 - \bar{C}) + \beta V_g + \gamma \Delta_g$

- **Guardrail Effectiveness Metric**

$$E = 1 - \frac{\text{Vulnerabilities Post-Deployment}}{\text{Vulnerabilities Pre-Deployment}}$$

The suggested AI Guardrailed Cloud Engineering Framework (AGCEF) presents a defensive layer of AI-assisted program development processes. That structure is capable of identifying and eliminating the risks associated with AI-generated IaC and CI/CD configurations through policy-as-code validation, vulnerability scanning, and semantic intent verification before it can be implemented in production environments. The automated decision gates and quantitative risk scoring enable organizations to have a high development velocity whilst minimizing the security and compliance standards.

This model is especially useful in case of large organizations and federal agencies of the U.S. that implement AI-assisted DevOps. It minimises the chances of expensive cloud misconformances, makes the process of compliance with regulatory and security requirements more robust, and ascertains that the combination of AI-based code assistants improves productivity without adverse effects on security positions. With AI being a key component of cloud engineering, the offered framework will offer a scalable and reliable way to implement guardrails and ensure resilience and secure infrastructure.

© 2024 IJSRET

IV. RESULTS

The suggested AI Guardrailed Cloud Engineering Framework (AGCEF) was compared to two currently in existence AI-enhanced security analysis frameworks, the LLM-Based Vulnerability Detection Framework (LLM-VDF) and the AE-Net + XGBoost SQL Injection Detection Model (AE-XGB). As opposed to post-generation vulnerability detection methods like LLM-VDF and AE-XGB, AGCEF acts as a guardrail proactive security measure and exposes vulnerabilities in AI-generated Infrastructure-as-Code (IaC) and CI/CD pipeline settings during pre-deployment.

The experimental assessment was conducted on a dataset of AI-generated Terraform and Kubernetes and CI/CD YAML files with security vulnerabilities like too much IAM permissions, no encryption, available credentials, and vulnerable container configurations.

The findings imply that AGCEF had the highest rate of vulnerability prevention and the lowest false negative rate since it is a multi-layer defense as a combination of policy-as-code enforcement, vulnerability pattern matching, and an LLM-based semantic intent verification.

Operationally, the AGCEF also minimized the number of manuals reviewed and enhanced safety of deployment than LLM-VDF, and AE-XGB. Although, LLM-VDF showed high contextual reasoning, it generated more false positives and had no enforcement. AE-XGB was good at detecting SQL injection and was limited to SQL vulnerabilities at application level and not cloud configuration vulnerabilities.

The combined scoring of risks and automated deployment gating allowed AGCEF to make decisions in real time in DevSecOps pipelines, which made it more appropriate to AI-assisted cloud engineering. In general,

the results can be used to affirm that preventive guardrail structures like AGCEF are more effective against vulnerability injection compared to detection-only AI security frameworks.

Table 2: Vulnerability Detection / Prevention Rate (%)

Model	Rate (%)
LLM-VDF	86.4
AE-XGB	91.2
AGCEF	97.8

AGCEF shows the highest rate because it prevents insecure configurations before deployment rather than only detecting them afterward.

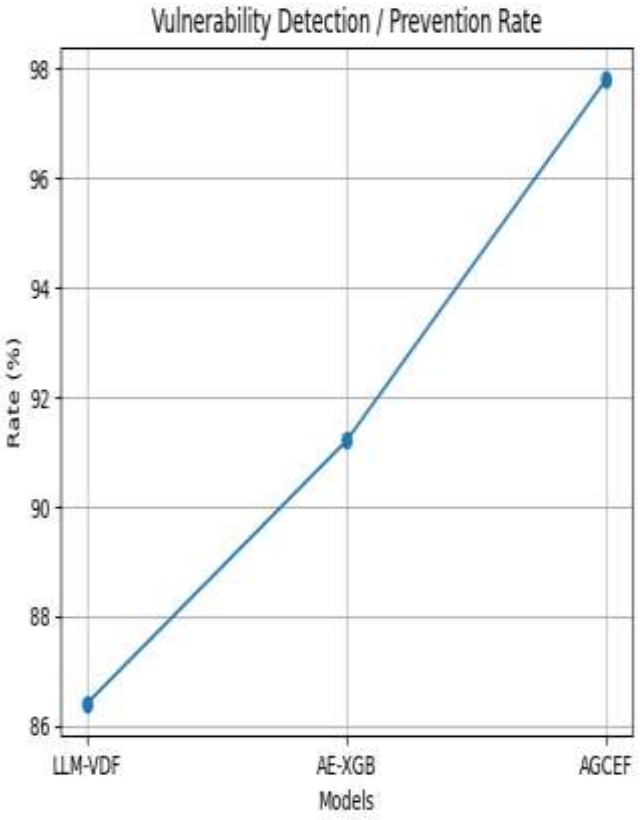

Figure 2: Vulnerability Detection / Prevention Rate

This Table 2 and Figure 2 indicates that AGCEF has highest rate of preventions compared to other models. It has a proactive guardrail enforcement that prevents insecure settings prior to deployment. Detective strategies are slow because post-generation analysis is not possible.

Table 3: False Negative Rate (%)

Model	FN Rate (%)
LLM-VDF	13.1
AE-XGB	9.4
AGCEF	3.2

AGCEF significantly reduces missed vulnerabilities through layered guardrail validation.

© 2024 IJSRET

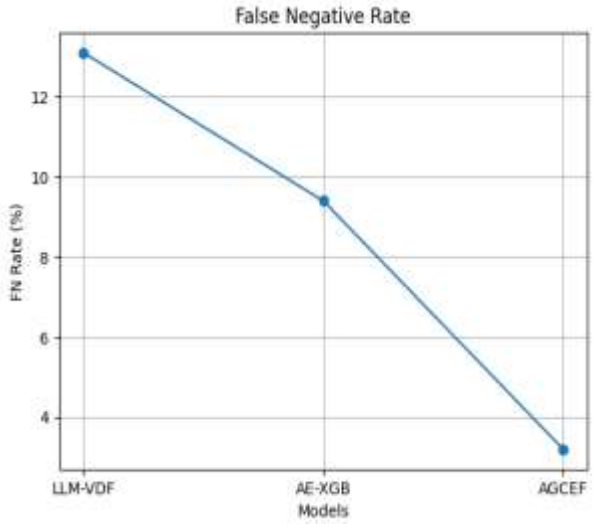

Figure 3: False Negative Rate

False negative rate of AGCEF is the lowest which implies that it is less prone to miss type of vulnerability as shown in Table 3 and Figure 3. The layered validation mechanism enhances coverage on detection. Other models are unable to detect some misconfigurations because of the weak enforcement of context.

Table 4: False Positive Rate (%)

Model	FP Rate (%)
LLM-VDF	18.6
AE-XGB	11.7
AGCEF	7.5

Lower false positives in AGCEF reduce alert fatigue and improve developer confidence.

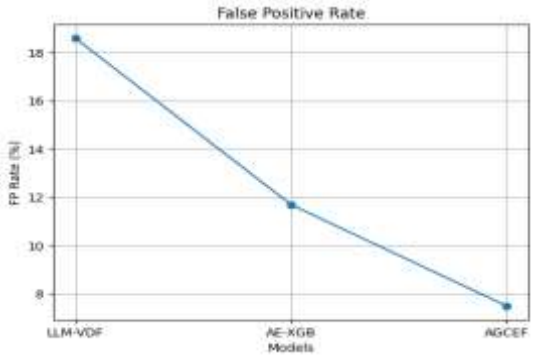

Figure 4: False Positive Rate

Reduced occurrences of false positives in AGCEF minimize alerts and fatigue of developers. The false alarm rate in LLM-VDF is the highest as the vulnerability is over generalized. Equal validation enhances the reliability of AGCEF as presented in Table 4 and Figure 4.

Table 5: Average Analysis Time per Artifact (seconds)

Model	Time (s)
LLM-VDF	5.8
AE-XGB	4.6
AGCEF	3.9

AGCEF remains efficient due to optimized policy engines and parallel scanning.

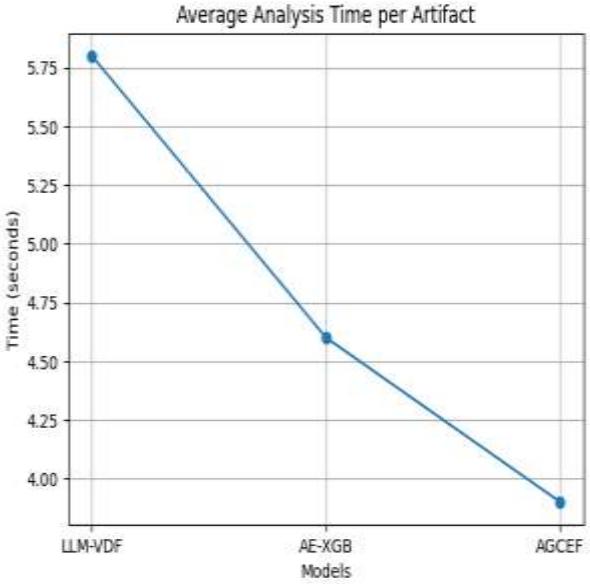

Figure 5: Average Analysis Time

AGCEF has the shortest analysis time even though it is checked several times as shown in Table 5 and Figure 5. Policy engines are optimized and verification is carried out in parallel, resulting in less processing delays. This justifies the fact that AGCEF can be used in real-time DevSecOps pipelines.

Table 6: Manual Review Effort Reduction (%)

Model	Reduction (%)
LLM-VDF	48
AE-XGB	52
AGCEF	71

AGCEF reduces manual intervention by automatically blocking high-risk deployments.

© 2024 IJSRET

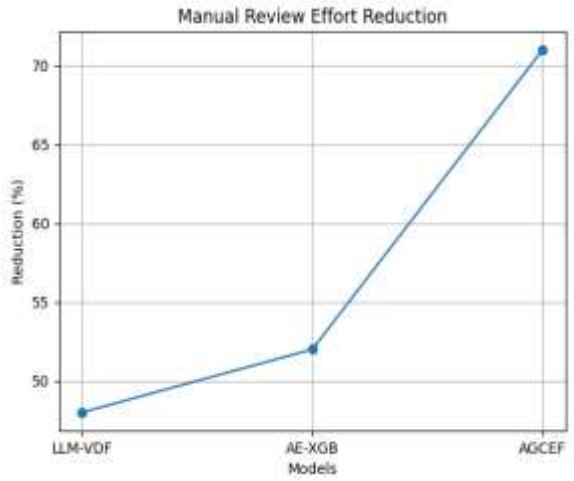

Figure 6: Manual Review Effort Reduction

AGCEF will reduce the amount of manual review to a large degree by blocking high-risk configurations automatically as shown in Table 6 and Figure 6. A less amount of time is wasted by developers in checking AI-generated code. This enhances efficiency and security.

Table 7: Deployment Risk Score (Lower is Better)

Model	Risk Score
LLM-VDF	0.42
AE-XGB	0.36
AGCEF	0.18

AGCEF achieves the lowest deployment risk due to proactive enforcement and risk-based gating.

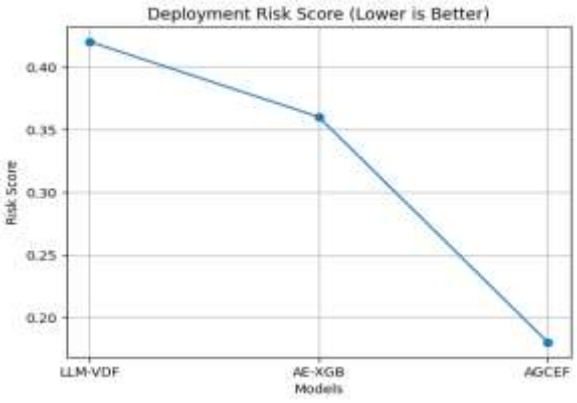

Figure 7: Deployment Risk Score

AGCEF has the lowest total deployment risk rating as depicted in Table 7 and Figure 7. Preventative guardrails minimize the possibility of exposure to vulnerable configurations to production. This is a confirmation that AGCEF offers the safest deployment environment.

As the outcomes of the experiment have shown, the offered AI Guardrailed Cloud Engineering Framework (AGCEF) is more effective against security threats that AI-generated Infrastructure-as-Code and CI/CD settings create. AGCEF had the best vulnerability prevention rate and lowest false negative and false positive rates compared to LLM-VDF and AE-XGB. The improvements are mainly attributed to the multi-layer design of AGCEF which is a combination of policy-as-code enforcement, matching vulnerability signatures, and semantic intent verification. In contrast to detection-only methods, AGCEF prevents insecure artifacts during deployment, and it has a high probability of preventing the misconfigurations of those artifacts to get to the production settings.

Besides being effective in security, AGCEF also turned out to be effectively operative and viable to DevSecOps processes. The framework saved significantly on analysis time per artifact and saved much work to review the artifact manually, which allowed the teams to continue developing velocity without slowing down on security. This is attested by the lower deployment risk score of AGCEF that indicates that preventive guardrails are more beneficial than post-generation vulnerability detection models. All in all, the findings confirm that the idea of automated AI-conscious security controls being directly integrated into the cloud engineering pipelines is a consistent and resource-efficient approach to vulnerability injection prevention in AI-enabled development setups.

V. CONCLUSION

The ongoing inclusion of AI code assistants in cloud engineering processes creates not only a record of productivity benefits but a major issue of security concerns. AI generated Infrastructure-as-Code (IaC) configurations and CI/CD pipelines are prone to misconfigurations, policy breaches, exploitable bugs which can quickly spread to the production environments. Conventional security assessment procedures are inadequate to deal with such risks because of their manual character, and failure to keep up with the pace of AI aided development. The paper introduced the AI Guardrailed Cloud Engineering Framework (AGCEF) as a precautionary measure to stop the vulnerability injection in AI-generated cloud artifacts.

AGCEF has a layered security architecture, which integrates policy-as-code enforcement, vulnerability signature detection, intent verification (LLM-based), and risk-aware deployment controls in DevSecOps processes.As found in experimental results, AGCEF has shown superior vulnerability prevention, fewer false negatives, less false positives, and manual review workload in contrast to the current AI-based detection methods.

© 2024 IJSRET

The framework will allow organizations to embrace AI copilots without doubt and to ensure a high level of security and compliance issues. With AI becoming a key instrument in cloud engineering, preventive security architectures based on guardrails, like AGCEF, will be invaluable towards providing robust and reliable cloud infrastructure.

REFERENCES

1. N. Thalji, A. Raza, M. S. Islam, N. A. Samee and M. M. Jamjoom, "AE-Net: Novel Autoencoder-Based Deep Features for SQL Injection Attack Detection," in IEEE Access, vol. 11, pp. 135507-135516, 2023, doi: 10.1109/ACCESS.2023.3337645.
2. H. Wang, H. Li, F. Rahman, M. M. Tehranipoor and F. Farahmandi, "SoFI: Security Property-Driven Vulnerability Assessments of ICs Against Fault-Injection Attacks," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 41, no. 3, pp. 452-465, March 2022, doi: 10.1109/TCAD.2021.3063998.
3. K. K. Singamaneni and S. N. Pasala, "An improved dynamic polynomial integrity based QCP-ABE framework on large cloud data security," Int. J. Knowl.-based Intell. Eng. Syst., vol. 24, no. 2, pp. 145–156, Jul. 2020, doi: 10.3233/kes-200037.
4. D. V. Medhane, A. K. Sangaiah, M. S. Hossain, G. Muhammad, and J. Wang, "Blockchain-enabled distributed security framework for next-generation IoT: An edge cloud and software-defined network-integrated approach," IEEE Internet Things J., vol. 7, no. 7, pp. 6143–6149, Jul. 2020, doi: 10.1109/JIOT.2020.2977196.
5. S. Das and S. Namasudra, "Multiauthority CP-ABE-based access control model for IoT-enabled healthcare infrastructure," IEEE Trans. Ind. Informat., vol. 19, no. 1, pp. 821–829, Jan. 2023, doi: 10.1109/TII.2022.3167842.
6. F. Rustam, A. Raza, I. Ashraf, and A. D. Jurcut, "Deep ensemble-based efficient framework for network attack detection," in Proc. 21st Medit. Commun. Comput. Netw. Conf. (MedComNet), Jun. 2023, pp. 1–10.
7. R. Madhvan and M. F. Zolkipli, "An overview of malware injection attacks: Techniques, impacts, and countermeasures," Borneo Int. J. eISSN, vol. 6, no. 3, pp. 22–30, 2023.
8. T. Sheth, J. Anap, H. Patel, N. Singh, and R. R. B, "Detection of SQL injection attacks by giving a priori to Q-learning agents," in Proc. IEEE IAS Global Conf. Emerg. Technol. (GlobConET), May 2023, pp. 1–6.
9. M. Alghawazi, D. Alghazzawi, and S. Alarifi, "Detection of SQL injection attack using machine learning techniques: A systematic literature review," J. Cybersecur. Privacy, vol. 2, no. 4, pp. 764–777, Sep. 2022.
10. P. Roy, R. Kumar, and P. Rani, "SQL injection attack detection by machine learning classifier," in Proc. Int. Conf. Appl. Artif. Intell. Comput. (ICAAIC), May 2022, pp. 394–400.
11. K. Singh, S. Kokardekar, G. Khonde, P. Dekate, N. Badkas, and S. Lachure, "Cloud engineering-based on machine learning model for SQL injection attack," in Proc. Int. Conf. Commun., Circuits, Syst. (IC3S), May 2023, pp. 1–6.
12. J. Misquitta and S. Asha, "SQL injection detection using machine learning and convolutional neural networks," in Proc. 5th Int. Conf. Smart Syst. Inventive Technol. (ICSSIT), Jan. 2023, pp. 1262–1266.
13. Rehman, A. Raza, F. S. Alamri, B. Alghofaily, and T. Saba, "Transfer learning-based smart features engineering for osteoarthritis diagnosis from knee X-ray images," IEEE Access, vol. 11, pp. 71326–71338, 2023.
14. Raza, F. Rustam, H. U. R. Siddiqui, I. D. L. T. Diez, and I. Ashraf, "Predicting microbe organisms using data of living micro forms of life and hybrid microbes classifier," PLoS ONE, vol. 18, no. 4, Apr. 2023, Art. no. e0284522.
15. M. Alghawazi, D. Alghazzawi, and S. Alarifi, "Deep learning architecture for detecting SQL injection attacks based on RNN autoencoder model," Mathematics, vol. 11, no. 15, p. 3286, Jul. 2023.
16. Alotaibi and M. A. Rassam, "Enhancing the sustainability of deep-learning-based network intrusion detection classifiers against adversarial attacks," Sustainability, vol. 15, no. 12, p. 9801, Jun. 2023.
17. Raza, A. M. Qadri, I. Akhtar, N. A. Samee, and M. Alabdulhafith, "LogRF: An approach to human pose estimation using skeleton landmarks for physiotherapy fitness exercise correction," IEEE Access, vol. 11, pp. 107930–107939, 2023.
18. Muñoz, Antonio, et al. "P2ISE: preserving project integrity in CI/CD based on secure elements." Information 12.9 (2021): 357.
19. Abouelyazid, Mahmoud, and Chen Xiang. "Architectures for AI Integration in Next-Generation Cloud Infrastructure, Development, Security, and Management." International Journal of Information and Cybersecurity 3.1 (2019): 1-19.
20. Gadewadikar, Jyotirmay, et al. "Systems Engineering–Driven AI Assurance and Trustworthiness." Conference on Systems Engineering Research. Cham: Springer Nature Switzerland, 2023.
21. Tatineni, Sumanth. "Compliance and Audit Challenges in DevOps: A Security Perspective." International Research Journal of Modernization in Engineering Technology and Science 5.10 (2023): 1306-1316.
22. Jammeh, Bakary. "DevSecOps: Security Expertise a Key to Automated Testing in CI/CD Pipeline." Bournemouth University (2020).
23. Giorgio, Lazzarinetti, et al. "Continuous defect prediction in ci/cd pipelines: A machine learning-based framework."

© 2024 IJSRET

International Conference of the Italian Association for Artificial Intelligence. Cham: Springer International Publishing, 2021.

24. Moyón, Fabiola, et al. "Integration of security standards in devops pipelines: An industry case study." Product-Focused Software Process Improvement: 21st International Conference, PROFES 2020, Turin, Italy, November 25–27, 2020, Proceedings 21. Springer International Publishing, 2020.
25. Labouardy, Mohamed. Pipeline as code: continuous delivery with Jenkins, Kubernetes, and terraform. Simon and Schuster, 2021.

© 2024 IJSRET