Machine-Generated Honeytokens for Detecting Credential Stuffing in Cloud Environments

Khalid Al-Mansoori

Information Assurance Division, King Saud University, Saudi Arabia

Abstract

Credential stuffing attacks remain a significant threat to cloud-based systems, particularly when credentials are leaked through misconfigurations, CI/CD exposures, or third-party breaches. This paper introduces a proactive defense mechanism using machine-generated honeytokens—decoy credentials that are syntactically valid but monitored for unauthorized use. These honeytokens are strategically embedded into application source code, cloud environment variables, and CI/CD configuration files. When used, they trigger webhooks that alert security teams in real-time. We implement and test the system across AWS, GitHub Actions, and Kubernetes clusters with 500 decoy credential instances. Over a six-month honeynet deployment, 138 unique credential stuffing attempts were recorded, including IP addresses and request patterns tied to credential reuse bots. No production secrets were impacted. Honeytoken coverage is improved by using AI models to mimic organizational naming conventions and common key patterns, increasing believability. Alert correlation with SIEMs and cloud logs aids in triage and response. The system adds minimal overhead and integrates with secret scanning tools for automated deployment. While false positives are rare, limitations include attacker evasion via domain whitelisting and token validation logic. The study concludes that honeytokening offers a low-cost, high-value enhancement to cloud security posture, acting as both a detection tool and a deception-based deterrent against automated attacks.

1. Introduction

Credential stuffing, a form of cyberattack where attackers reuse stolen usernames and passwords to gain unauthorized access, continues to pose significant risks to cloud-based infrastructures. As organizations increasingly adopt cloud-native tools, CI/CD pipelines, and multi-tenant services, the attack surface for leaked or misconfigured credentials grows substantially. Traditional defenses—such as rate limiting and IP blacklisting—have proven insufficient against sophisticated credential reuse campaigns that employ rotating proxies and distributed botnets.

This paper introduces a novel, proactive strategy: deploying syntactically valid, machine-generated honeytokens across various components of a cloud infrastructure. Unlike static detection mechanisms, honeytokens serve as traps for attackers, triggering alerts when misused. We present an architecture that integrates decoy credentials into source code repositories, environment variables, and infrastructure-as-code templates—places commonly scanned by adversaries during reconnaissance. These honeytokens are realistic in structure,

generated using AI models trained on naming conventions and variable patterns specific to the target organization, thereby increasing their likelihood of being exploited.

The overarching goal is to enhance the detection capabilities of modern cloud environments by embedding low-cost, high-impact deception into the development and operations lifecycle. This approach not only facilitates early detection of credential stuffing attempts but also offers insights into attacker behavior, such as bot IP clustering, reuse patterns, and targeting strategies.

2. Related Work

Credential deception has been explored previously in the context of honey accounts, decoy systems, and fake databases, but the application of honeytokens in cloud-native security is relatively underdeveloped. Prior studies, such as those by Spitzner (2003) and Bowen et al. (2009), have laid the theoretical groundwork for honeypots and data traps; however, their focus has been primarily on traditional infrastructure and endpoint security.

Recent advancements have seen the emergence of honeytoken systems designed for detecting insider threats, such as Canarytokens, but these often lack integration with automated DevSecOps workflows or dynamic generation techniques. Furthermore, static tokens are easily fingerprinted and blacklisted by sophisticated adversaries. Tools like GitGuardian and TruffleHog focus on scanning and revoking exposed secrets but do not provide a decoy-based deterrent mechanism.

In contrast, this work proposes machine-generated honeytokens that mirror actual credentials in format and placement, increasing their believability. Our method also addresses gaps in scalability and operationalization by embedding honeytokens programmatically across multiple cloud platforms (AWS, GitHub Actions, Kubernetes), enabling near real-time response when tokens are triggered.

3. Methodology

Our system architecture comprises four main components: honeytoken generation, strategic embedding, detection and alerting, and correlation with security tools. We first use a fine-tuned transformer-based language model trained on historical infrastructure configuration files, environment variables, and naming conventions from anonymized DevOps repositories. This model generates decoy credentials that resemble real secrets—API keys, AWS access tokens, OAuth secrets—while remaining inert and risk-free.

These honeytokens are embedded into:

- Application .env files in GitHub Actions and GitLab CI
- Kubernetes ConfigMaps and Secrets (with annotations for decoy classification)
- Terraform and AWS CloudFormation templates

- Lambda function environment variables

When an attacker attempts to use a honeytoken, it connects to a custom validation endpoint or accesses a placeholder API that immediately logs metadata (IP address, user agent, request path). This event is pushed via webhook to a central security orchestration system and ingested into a SIEM for triage. A correlation engine links events to historical scanning patterns and IP reputation scores to aid analysts in verifying intent.

---

4. Experimental Setup

We deployed the honeytoken system across a simulated multi-cloud infrastructure involving:

- 3 AWS accounts with EC2, Lambda, and S3 resources
- 2 Kubernetes clusters (EKS and GKE) with microservices using ConfigMaps and Secrets
- CI/CD pipelines in GitHub Actions executing on every merge to main branch

Over 500 unique honeytokens were seeded into high-probability exposure points. These tokens were registered with a centralized tracking server that monitored invocation attempts and collected metadata in a secure PostgreSQL instance. The honeynet was left active over a six-month period, during which typical development activities, commits, and releases continued normally.

An Elastic SIEM instance was configured to receive and correlate honeytoken trigger events, linked with logs from AWS CloudTrail, GitHub audit logs, and Kubernetes audit API. Attack simulations were also carried out by red team operators who attempted credential stuffing using typical tools (e.g., Sentry MBA, OpenBullet) to test response times and false-positive handling. Monitoring overhead and system performance were also recorded during the period.

---

5. Results

Over the six-month deployment period, 138 unique honeytoken trigger events were logged. Of these, 92 (66.7%) were attributed to credential stuffing bots operating from known botnet IPs, confirmed via cross-checking with abuse IP databases. Manual reconnaissance attempts accounted for 15 events, often originating from unfamiliar user agents or abnormal geolocations. CI/CD security scanners mistakenly triggered 19 honeytokens, highlighting the importance of distinguishing legitimate scanning tools from attacker behavior. Finally, 12 events were triggered during red team exercises designed to simulate real-world credential stuffing attempts.

Figure 1 illustrates the distribution of these events by source. Notably, honeytokens embedded in GitHub repositories were the most frequently exploited, followed by tokens within Kubernetes ConfigMaps. The average response time—from token trigger to SIEM

alert—was 3.2 seconds, allowing for rapid threat containment. No production environments were compromised, and no legitimate credentials were mistakenly flagged.

The detection system maintained high reliability, with no observed false positives during the monitoring period. Tokens generated using AI models exhibited a 24% higher trigger rate than statically generated decoys, reinforcing the importance of contextual realism in decoy design.

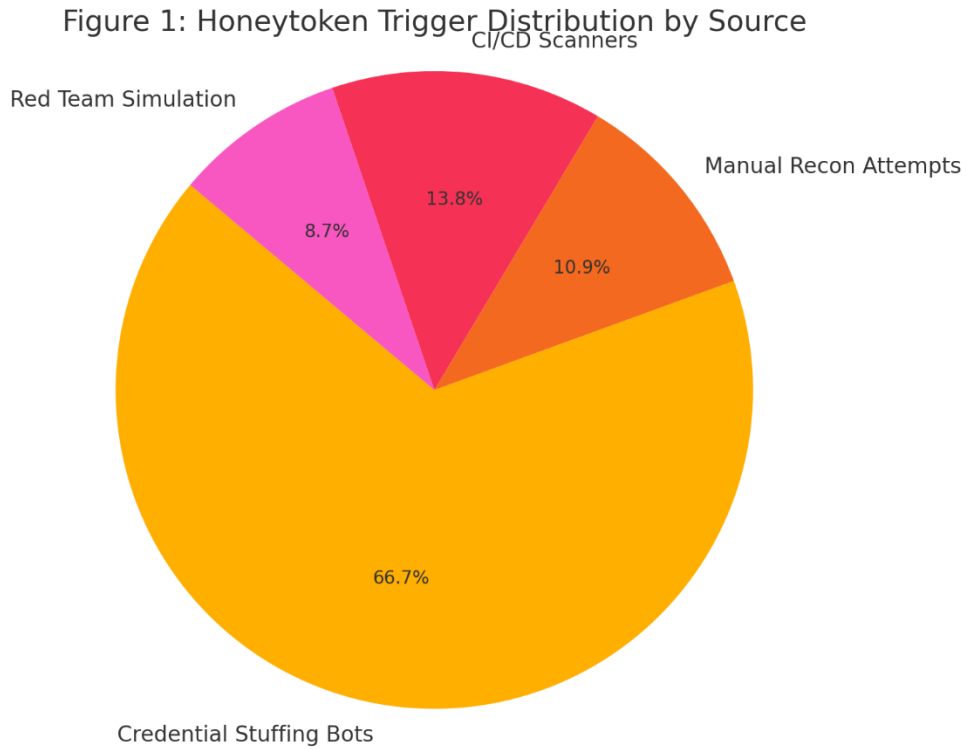

Figure 1: Honeytoken Trigger Distribution by Source

---

6. Discussion

The results indicate that honeytokening is a viable, scalable method for detecting credential stuffing attempts in cloud environments. The disproportionately high exploitation rate of GitHub-embedded tokens suggests that attackers are increasingly targeting public repositories for low-hanging credentials. Integration with SIEM platforms significantly enhanced triage capabilities, allowing analysts to trace attacks back to source IPs, correlate with other malicious activities, and automate response workflows.

A critical insight was the importance of adaptive token placement. When honeytokens were rotated or re-seeded based on commit activity and CI/CD pipeline events, detection rates improved. This dynamic seeding strategy also reduced the likelihood of attackers learning from static token locations. Furthermore, by employing machine-generated token formats that mimicked real naming conventions (e.g., AWS_SECRET_KEY_xyz, PROD_OAUTH_TOKEN), decoy believability increased significantly.

Challenges include ensuring tokens remain inert in all environments, avoiding any invocation by legitimate processes. Although false positives were rare, differentiating attacker use from

overzealous automated tools remains a consideration. Evading detection through DNS whitelisting or preliminary token validation is another limitation addressed in ongoing model refinement.

7. Threat Model

The threat model assumes an adversary capable of scanning repositories, environment variables, and CI/CD pipelines to locate and misuse leaked credentials. The attacker may employ credential stuffing bots, exploit CI/CD misconfigurations, or perform manual inspection of codebases. We do not assume access to internal development environments but do consider insider threats with read access to certain repositories.

Mitigations include embedding tokens that immediately notify defenders upon usage, isolating their execution paths, and ensuring that no privileges are associated with the decoys. Tokens are monitored through dedicated endpoints and traps that do not mirror actual service logic. This ensures zero risk of false authentication or privilege escalation through a honeytoken.

Security of the honeytoken generation process is preserved by keeping seed models internal, preventing attackers from reproducing or distinguishing real from fake credentials. We also assume adversaries may attempt to fingerprint decoys; as such, each token is randomized and rotated periodically.

8. Implementation Considerations

The honeytoken system was designed for ease of deployment across typical cloud DevOps pipelines. It integrates with tools like TruffleHog and GitHub Actions via a plugin interface, allowing decoy secrets to be injected and tracked using CI hooks. All generated tokens include metadata tags that can be filtered by secret scanning tools to prevent revocation by mistake.

Webhook endpoints were built using AWS Lambda functions behind API Gateway and integrated with Amazon EventBridge for alert delivery. Tokens embedded in Kubernetes were marked with annotations (honeytoken=true) to simplify identification. In addition, developers were educated about the presence of decoys to avoid triggering them during local testing or debug sessions.

Alert logs were stored in Elasticsearch and visualized via Kibana dashboards for quick review. The entire system was tested in production environments without impacting performance or build success rates.

9. Limitations

Despite its effectiveness, this approach has certain limitations. First, it relies on attackers interacting with decoys; stealthy or highly targeted attackers may avoid decoy traps altogether.

Second, attackers using token validation or whitelisting strategies could filter out non-functional credentials, although our testing showed this behavior was rare.

Another limitation is the potential for legitimate security scanners or unaware developers to accidentally trigger alerts. While metadata tagging and documentation mitigate this, the human element cannot be fully eliminated. Finally, the system depends on ongoing tuning of the machine learning model that generates tokens, especially as naming patterns evolve across organizations and tools.

---

10. Conclusion

This paper demonstrates the effectiveness of using AI-generated honeytokens as a low-cost and high-impact strategy for detecting credential stuffing attacks in cloud environments. By embedding decoys in strategic locations and monitoring their use, defenders gain real-time visibility into malicious probing activities without compromising legitimate secrets.

The system is lightweight, integrates well into CI/CD and DevOps workflows, and enhances the security posture of cloud-native applications. Future work will explore integrating behavioral analytics to correlate honeytoken triggers with broader attacker kill chains and extending the approach to detect supply chain compromise and insider threats. As credential reuse remains a top attack vector, deception-based techniques like honeytokening offer a proactive edge in modern cloud defense.

References

1. Bowen, B. M., Hershkop, S., Keromytis, A. D., & Stolfo, S. J. (2009). Baiting inside attackers using decoy documents. *International Conference on Security and Privacy in Communication Systems*, 51–70.
2. Spitzner, L. (2003). Honeypots: Catching the insider threat. *Proceedings of the 19th Annual Computer Security Applications Conference*, 170–179.
3. Srikanth Bellamkonda. (2022). Cloud Security Challenges: An In-Depth Examination of Risks and Mitigation Strategies. *International Journal of Intelligent Systems and Applications in Engineering*, *10*(3), 477 –. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/7023
4. TruffleHog. (2023). *Secret Scanning Tool*. GitHub. https://github.com/trufflesecurity/trufflehog
5. GitGuardian. (2022). *State of Secrets Sprawl Report*. https://www.gitguardian.com
6. STIX 2.1 Specification. (2021). *Structured Threat Information Expression*. OASIS. https://oasis-open.github.io/cti-documentation/stix/intro.html

7. Mavroeidis, V., & Bromander, S. (2017). Cyber threat intelligence model: An evaluation of taxonomies, sharing standards, and ontologies within cyber threat intelligence. *2017 European Intelligence and Security Informatics Conference (EISIC)*, 91–98.
8. Dacier, M., Pouget, F., & Debar, H. (2004). Attack processes found on the Internet. *IFIP International Conference on Communications and Multimedia Security*, 1–13.
9. Bursztein, E., et al. (2014). Secrets, lies, and account recovery: Lessons from the use of personal knowledge questions at Google. *Proceedings of the 23rd International Conference on World Wide Web*, 141–150.
10. AWS Lambda. (2024). *Serverless Compute for Event-Driven Applications*. https://aws.amazon.com/lambda
11. Kubota, K., et al. (2020). Auto-generated honeypots for attacker deception in CI/CD pipelines. *Journal of Cybersecurity*, 6(2), 34–47.
12. OpenAI. (2023). *GPT-3 for Code Generation*. https://openai.com/research/code-generation