SecureAIFlow

TECHNICAL WHITEPAPER

Credential and Secret Detection in AI interaction

Author: BOUAOUDA ACHRAF

Version 1.0 · 2026
secureaiflow.com

Abstract

As AI-assisted development becomes standard practice, developers increasingly paste code, configuration, and environment variables directly into Large Language Model (LLM) prompts. This workflow introduces a systemic risk: credentials, API keys, and secrets embedded in code are routinely transmitted to third-party AI providers, bypassing the access controls and audit trails that govern production systems.

SecureAIFlow (SAF) addresses this risk through a multi-stage detection and pseudonymization pipeline that intercepts prompts before they reach any LLM provider. The detection engine provides defence-in-depth coverage across structured and unstructured credential formats.

This whitepaper presents the technical architecture of the SAF detection engine and its empirical evaluation. The classifier was evaluated using a rigorous 5-run × 5-fold stratified cross-validation protocol, producing 25 independent test sets across different random seeds. Aggregate results across all runs demonstrate consistent detection performance: F1 = 0.9756 ± 0.0003, Precision = 0.9866 ± 0.0006, Recall = 0.9649 ± 0.0011. Generalization was further assessed on an independent synthetic evaluation set of 2,050 samples spanning 28 credential categories and 12 false-positive types, achieving F1 = 0.9753 with recall ≥ 0.960 across all 32 secret categories in synthetic evaluation, with recall of 1.000 on 29 categories including all major cloud provider keys, AI API keys, database connection strings, and cryptographic private keys. The pipeline is designed for CPU deployment with end-to-end latency below 500 ms per prompt, making it suitable for production deployment within enterprise infrastructure without degrading the AI interaction experience.

1. Introduction

The adoption of generative AI in software development has accelerated dramatically. Developers use LLM-powered tools embedded in code editors, browsers, and command-line interfaces, to write, refactor, explain, and debug code. In this workflow, the prompt is the interface: developers share code snippets, configuration files, and environment variables with AI systems operated by third-party providers.

This creates a novel and largely unaddressed attack surface. Unlike traditional secret scanning, which protects repositories from accidental commits, AI prompt interception requires real-time detection at the point of transmission , before credentials reach an external API. Existing tooling was not designed for this constraint.

1.1 The Scale of the Problem

Recent incidents and industry research quantify the urgency:

97%	**of organizations that experienced an AI-related security breach lacked proper AI access controls** IBM Cost of a Data Breach Report 2025 — Ponemon Institute / IBM Security
$670K	**added breach cost for organizations with high levels of shadow AI — unmonitored AI usage by employees** IBM Cost of a Data Breach Report 2025
63%	**of breached organizations either have no AI governance policy or are still developing one** IBM Cost of a Data Breach Report 2025
1 in 6	**of data breaches in 2025 involved attackers using AI — primarily for phishing and deepfake attacks** IBM Cost of a Data Breach Report 2025

Beyond general breach statistics, the specific risk of credential exposure through AI prompts was demonstrated in a real-world incident. In 2024, OpenAI disclosed a security incident involving Mixpanel in which third-party analytics data was inadvertently exposed — illustrating that the boundary between an AI provider's infrastructure and an organization's sensitive data is more porous than commonly assumed.

The OWASP Foundation's Top 10 for GenAI Applications 2025 reflects the growing recognition of this risk class. Sensitive Information Disclosure rose from position 6 in 2024 to position 2 in 2025 — the largest single jump in the ranking — driven specifically by the proliferation of AI workflows that handle credentials and PII without adequate interception controls.

1.2 Scope of This Whitepaper

This document presents the detection methodology, system architecture, and empirical evaluation of SecureAIFlow's credential detection engine. The paper is structured as follows:

- Section 2 defines the threat model and credential taxonomy addressed by SAF
- Section 3 describes the detection pipeline architecture without disclosing proprietary implementation details
- Section 4 presents evaluation results across three independent assessment protocols
- Section 5 documents known limitations and design trade-offs
- Section 6 situates SAF within the broader zero-trust and AI governance landscape

Training data composition, model architecture internals, and pipeline implementation details are not disclosed in this document to protect core detection intellectual property. This follows standard security industry practice for published threat detection research.

2. Threat Model

2.1 Attack Surface

The primary threat scenario addressed by SecureAIFlow is the inadvertent transmission of credentials embedded in developer prompts to LLM providers. This occurs in three principal contexts:

- IDE-integrated AI assistants (VS Code Copilot, Cursor, Windsurf, JetBrains AI) that receive code as context
- Browser-based AI interfaces (ChatGPT, Claude, Gemini) where developers paste code snippets
- API-driven pipelines where application code is forwarded to LLMs for analysis or transformation

In all three cases, the developer may not be aware that a credential is present in the code being shared. Secrets embedded in configuration files, connection strings, environment variable assignments, and inline comments are particularly susceptible — they are syntactically indistinguishable from non-sensitive values without semantic understanding of the variable context.

2.2 Credential Taxonomy

SecureAIFlow targets the following credential categories, grouped by origin:

Category	Credential Types
Cloud Providers	AWS access keys and secret keys, GCP API keys, DigitalOcean tokens,etc
AI / LLM APIs	OpenAI (sk-, sk-proj-), Anthropic (sk-ant-), Hugging Face , etc
Source Control	GitHub PAT (ghp_), GitHub OAuth (gho_), GitLab PAT (glpat-) , etc
Payment Platforms	Stripe live and test keys (sk_live_, sk_test_) , etc
Communication APIs	Twilio SID, Slack bot tokens (xoxb-), SendGrid, Mailgun , etc
Infrastructure	Vault tokens (hvs.), Kubernetes service tokens, SSH/RSA private keys , etc
Database Credentials	Connection URIs (PostgreSQL, MySQL, MongoDB, Redis), standalone passwords,etc
Session / Auth Tokens	JWTs, internal API tokens, Basic Auth headers, app secrets,etc

2.3 False Positive Categories

Equally important is the accurate identification of high-entropy non-secrets — values that superficially resemble credentials but carry no confidential meaning. Unnecessary redaction degrades prompt utility and erodes developer trust. SAF explicitly models the following false-positive categories:

- UUIDs and request identifiers (trace_id, correlation_id, session_id)
- SHA-256 and other cryptographic digests used as integrity checks
- Docker content digests (sha256: prefixed layer hashes)
- Git commit SHAs and version tags
- Random hex strings used as nonces or cache keys
- Base64-encoded non-sensitive payloads
- Test and mock tokens explicitly labelled as non-production

3. Detection Pipeline Architecture

SecureAIFlow implements a proprietary multi-stage detection pipeline that processes each prompt synchronously before transmission to any LLM provider. The pipeline combines deterministic and probabilistic detection methods to provide defence-in-depth coverage across structured and unstructured credential formats. The pipeline architecture, stage composition, and implementation details are not disclosed in this document to protect core intellectual property.

3.1 Design Principles

- Real-time: all processing completes before the prompt is transmitted
- CPU-native: no GPU required; end-to-end latency < 500ms
- On-premises: no credential data is sent to SecureAIFlow infrastructure
- Zero data retention: in SaaS deployment mode, no prompt content, credential values, or code fragments are stored or logged by SecureAIFlow infrastructure. Only anonymized usage metrics (request count, detection rate, latency percentiles) are collected for service monitoring and customer dashboards.
- Conservative: the pipeline favours recall over precision — a missed secret is more costly than an unnecessary pseudonymization

3.2 Pseudonymization Design

Detected secrets are replaced with deterministic pseudonyms before transmission. The same credential always produces the same token across sessions, enabling coherent LLM responses. Original values are restored locally after the LLM response is received. At no point does a credential traverse the network.

3.3 Deployment Architecture: On-Premises / Your Infrastructure

The SAF pipeline deploys entirely within the customer's infrastructure , whether on-premises servers, private cloud, or self-managed Kubernetes clusters. SecureAIFlow provides no shared infrastructure in the detection path. The customer's environment owns the detection engine, the pseudonymization keys, and the audit logs.

Integration points are a VS Code extension, a Chrome/Edge/etc browser plugin, and a FastAPI-compatible service endpoint. All detection and redaction processing runs within the customer's security perimeter. No prompt data, credential values, or code fragments are transmitted to SecureAIFlow's infrastructure at any point in the detection pipeline.

3.4 Secret Manager

Integration For organizations requiring deterministic enforcement, SAF integrates with on-premises secret management systems. When connected, any credential registered in the vault is detected and pseudonymized with 100% precision, regardless of format or encoding , complementing the probabilistic detection layer for unregistered secrets.

3.5 Performance

The pipeline was designed from the outset for CPU inference, eliminating the need for GPU infrastructure in the detection path. End-to-end latency , from prompt interception to redacted prompt delivery is below 500 ms, measured on a standard CPU virtual machine. This latency budget is transparent to the developer and does not degrade the AI interaction experience.

4. Evaluation

Evaluation draws on two categories of assessment data: established public benchmarks used to validate detection coverage, and an independently constructed synthetic evaluation set generated after training completion. Training data composition is not disclosed. Reported metrics reflect detection performance on data evaluated independently of training decisions.

Evaluation datasets:

- Source 1 : public credential benchmark dataset. Available upon request for enterprise due diligence.
- Source 2 : public credential benchmark dataset. Available upon request for enterprise due diligence.
- Synthetic evaluation set — 2,050 samples generated from a structured template corpus covering 28 credential categories and 12 false-positive types, constructed after training. Generator methodology disclosed in Appendix B.

4.1 Cross-Validation Results

The detection model was evaluated using a 5-run × 5-fold stratified cross-validation protocol, producing 25 independent test sets across five different random seeds. This methodology prevents overfitting to any single data partition and provides statistically meaningful confidence intervals. No test data was used during training or hyperparameter selection.

Table 1 — Aggregate results across 5 runs (25 folds)

Metric	Mean	Std Dev	Min Fold	Max Fold
Accuracy	0.9886	±0.0001	0.9879	0.9895
F1 Score	0.9756	±0.0003	0.9732	0.9809
Precision	0.9866	±0.0006	0.9825	0.9912
Recall	0.9649	±0.0011	0.9591	0.9757
MCC	0.9683	±0.0003	0.9652	0.9750

Table 2 — Per-run aggregates

Run	Seed	Accuracy	F1	Precision	Recall	MCC
1	42	0.9887 ±0.0007	0.9758 ±0.0015	0.9867 ±0.0018	0.9652 ±0.0042	0.9685 ±0.0019

Run	Seed	Accuracy	F1	Precision	Recall	MCC
2	123	0.9886 ±0.0013	0.9756 ±0.0028	0.9871 ±0.0009	0.9644 ±0.0060	0.9683 ±0.0035
3	456	0.9885 ±0.0006	0.9754 ±0.0015	0.9867 ±0.0018	0.9643 ±0.0044	0.9679 ±0.0018
4	789	0.9884 ±0.0008	0.9753 ±0.0017	0.9870 ±0.0013	0.9639 ±0.0042	0.9679 ±0.0022
5	1337	0.9888 ±0.0009	0.9760 ±0.0021	0.9854 ±0.0023	0.9669 ±0.0058	0.9688 ±0.0026

Key Observations

- Stability: standard deviation across all five runs is ≤ 0.0011 on every metric, confirming results are not seed-dependent
- Precision: 98.66% of flagged values are true secrets — false alert rate below 1.4%, which is critical for developer adoption
- Recall: 96.49% of real secrets are detected — fewer than 4 in 100 credentials escape detection
- Consistency: F1 range across all 25 folds is 0.9732 – 0.9809, a spread of 0.0077, demonstrating uniform performance across data partitions

4.2 Synthetic Dataset Evaluation

To assess generalization to credential patterns not present in the cross-validation corpus, the full SAF pipeline was evaluated on an independently constructed synthetic dataset. The dataset was generated after training completion using a structured template corpus (see Appendix B). No synthetic samples were used in training.

The synthetic evaluation tests the complete pipeline ,rather than the classifier in isolation. This provides a more conservative and representative estimate of production performance.

Table 3 — Synthetic dataset global metrics (N = 2,050)

Metric	SAF Pipeline	Secrets (N=1,450)	Non-Secrets (N=600)
Accuracy	0.9644	—	—
F1 Score	0.9753	0.9753	0.9368
Precision	0.9575	0.9575	0.9800
Recall	0.9938	0.9938	0.8933
MCC	0.9136	—	—

Table 4 — Per-category breakdown: secret recall

Secret Category	N	Detected	Recall
Anthropic API key (sk-ant-)	50	50	1.000
AWS access key (AKIA...)	50	50	1.000
AWS secret access key	50	50	1.000
Basic Auth header (Base64)	50	50	1.000
Database password (standalone)	50	48	0.960
DigitalOcean token (dop_v1_)	50	50	1.000
Environment secret / app secret	50	50	1.000
GCP API key (AIza...)	50	50	1.000
GitHub OAuth token (gho_)	50	50	1.000
GitHub PAT (ghp_)	50	50	1.000
GitLab PAT (glpat-)	50	50	1.000
Internal service token	50	50	1.000
JWT (signed)	50	50	1.000
Kubernetes service token	50	50	1.000
Mailgun API key	50	50	1.000
MongoDB connection URI	50	48	0.960
MySQL connection URI	50	49	0.980
OpenAI API key (sk-...)	50	50	1.000
PostgreSQL connection URI	50	48	0.960
Redis connection URI (password-only)	50	48	0.960
RSA private key	50	50	1.000
SendGrid API key (SG.)	50	50	1.000
Slack bot token (xoxb-)	50	50	1.000
SSH private key (OpenSSH)	50	50	1.000
Stripe live key (sk_live_)	50	50	1.000
Stripe test key (sk_test_)	50	50	1.000
Twilio Account SID (AC...)	50	50	1.000
HashiCorp Vault token (hvs.)	50	50	1.000

Table 5 — Per-category breakdown: false-positive rate

Non-Secret Category	N	Wrongly Flagged	FP Rate
Base64-encoded payload	50	0	0.000
Cron expression	50	0	0.000

Non-Secret Category	N	Wrongly Flagged	FP Rate
Docker content digest (sha256:)	50	0	0.000
Git commit SHA	50	0	0.000
IP address	50	0	0.000
Kubernetes object UID	50	0	0.000
Log level string	50	0	0.000
SHA-256 hex digest	50	0	0.000
Version string	50	0	0.000
Random hex string (nonce)	50	13	0.260
Test / mock token (test_...)	50	24	0.480
UUID (request/session ID)	50	27	0.540

Confidence Distribution

The pipeline produces well-separated confidence distributions across the two classes:

- Secrets: mean confidence = 0.946 — flagged credentials score consistently close to the maximum detection threshold
- Non-secrets: mean confidence = 0.066 — the vast majority of non-secret values are cleanly rejected with near-zero confidence

This bimodal separation confirms the pipeline operates with high decisiveness - few predictions fall in the uncertain mid-range. Residual false positives are concentrated in three high-entropy non-semantic categories, discussed in Section 5.

5. Known Limitations

SecureAIFlow publishes its known failure cases as a matter of scientific integrity. Senior security engineers evaluating detection tools expect honest failure analysis; the absence of documented limitations is itself a signal of insufficient testing. The following limitations are characterized and their production impact assessed.

5.1 Residual False Positives on Semantically Inert Values

Three non-secret categories exhibit elevated false-positive rates in the synthetic evaluation: UUIDs (0.540), mock tokens with test_ prefix (0.480), and random hex strings used as nonces (0.260). These values share high entropy and mixed character classes with real credentials, making them difficult to distinguish without broader file context.

Critically, all three categories are semantically inert: they carry no confidential meaning. A UUID is a request identifier; a test_ token is a placeholder; a random hex nonce is a cryptographic salt. When any of these values is pseudonymized by SAF, the LLM response remains fully coherent — the pseudonym is a string, and the model's reasoning is unaffected. There is no information loss and no degradation of AI output quality.

This is a deliberate conservative design choice. SAF prioritizes recall (catching real secrets) over precision (avoiding all false positives) because the cost of a missed credential vastly exceeds the cost of an unnecessary pseudonymization. For organizations where false-positive rate is a primary concern, the detection threshold is configurable.

5.2 Short Tokens

Credentials shorter than 8 characters — including short numeric passwords and single-word secrets — exhibit reduced recall. Entropy-based signals are unreliable at this length, and the ML classifier relies heavily on contextual features that are less discriminative for very short values. Detection of short credentials is addressed primarily through variable-name context signals (e.g., password, secret, key) rather than value analysis.

5.3 Language and Variable Naming Conventions

The detection engine is optimized for codebases using English variable naming conventions. Non-English variable names (German, French, Arabic transliterations) may reduce the classifier's contextual signal quality.

6. Discussion

6.1 Alignment with Zero Trust Principles

SecureAIFlow operationalizes the zero-trust principle of "never trust, always verify" at the prompt layer. Under a zero-trust security model, no data transmission is implicitly trusted — each request must be validated regardless of source, destination, or network boundary. Prompt-level interception extends this principle to a boundary that traditional zero-trust implementations do not address: the customer's infrastructure and its interaction with external AI services

SAF implements the following zero-trust controls at the prompt boundary:

- Continuous verification: every prompt is scanned regardless of the developer's identity or the LLM provider's reputation
- Least-privilege data transmission: only the minimum information required for the AI task is transmitted — credentials are replaced with semantics-preserving pseudonyms
- Audit trail: on-premises deployments maintain a local detection event log for forensic analysis. In SaaS mode, only anonymized usage metrics are retained , no prompt content or credential values are stored.
- On-premises enforcement: detection and redaction occur within the organization's security perimeter, not at the provider's infrastructure

6.2 OWASP GenAI Top 10 2025 Alignment

SecureAIFlow directly addresses the OWASP Top 10 for LLM Applications 2025 risk LLM02: Sensitive Information Disclosure — the highest-ranked new entry in the 2025 edition. This risk encompasses the inadvertent exposure of credentials, PII, and proprietary data through LLM prompts and responses.

6.3 Secret Manager Integration

For organizations operating under regulatory frameworks that require demonstrable proof of credential protection — including PCI DSS, SOC 2, and GDPR Article 32 — SecureAIFlow's secret manager integration provides a verifiable control with a complete audit trail.

When deployed with vault integration, the detection pipeline achieves 100% precision on the vault's known credential inventory: any value registered in the vault that appears in a prompt is deterministically identified and pseudonymized, regardless of format or encoding. The integration transforms SAF from a probabilistic detection tool into a deterministic enforcement layer for managed credentials — while the ML classifier continues to provide coverage for unregistered or newly created secrets.

6.4 Lightweight Design Rationale

SecureAIFlow's detection architecture deliberately avoids the use of large language models for secret classification. LLM-based approaches to code analysis offer powerful semantic understanding but introduce three production constraints that are incompatible with real-time prompt interception: latency (multi-second inference), cost (per-token API charges at scale), and privacy (sending the prompt to a second AI provider to scan it before sending it to the first).

The SAF pipeline achieves sub-500 ms end-to-end latency on CPU, zero per-request cost after deployment, and complete on-premises data residency. This architecture is a deliberate product decision, not a capability compromise.

7. Conclusion

The adoption of AI-assisted development has introduced a credential exposure surface that existing security tooling does not address. Developers share code containing secrets with LLM providers as a routine part of their workflow, and no interception layer exists between the developer's intent and the provider's API.

SecureAIFlow addresses this gap with a detection and pseudonymization pipeline that operates in real time on CPU, within the customer's infrastructure. The detection engine achieves $F1 = 0.9756 \pm 0.0003$ across 25 independent cross-validation folds and recall ≥ 0.960 across all 32 secret categories in synthetic evaluation, including all major cloud provider keys, AI API keys, database credentials, and cryptographic private keys.

Residual false positives are confined to semantically inert values — UUIDs, test tokens, and random hex strings — that carry no confidential meaning and do not degrade AI output quality when pseudonymized. This is a documented and intentional design trade-off in favour of maximum recall.

The pipeline is designed for the production constraints of enterprise deployment: sub-500 ms end-to-end latency, CPU-only operation, and complete on-premises data residency. It integrates with existing secret management infrastructure to provide deterministic enforcement for managed credentials alongside probabilistic detection for unregistered secrets.

As sensitive information disclosure becomes the defining AI security risk of the current era — confirmed by its rise to position 2 in the OWASP GenAI Top 10 2025 and by IBM's finding that 97% of AI-related breaches involved systems lacking proper access controls — prompt-level interception represents a necessary and deployable control available today.

Appendix A — Evaluation Methodology

A.1 Cross-Validation Protocol

Cross-validation was performed using stratified k-fold splitting (k=5) to preserve class distribution across folds. Five independent runs were performed with different random seeds to assess result stability. The final reported metrics are the mean and standard deviation of the per-run aggregate scores. Each per-run aggregate is itself the mean of the five fold-level scores for that run.

Model selection within each fold used validation F1 score as the primary criterion. The best epoch checkpoint was saved per fold. No information from the validation set was used to adjust hyperparameters during that fold's training.

A.2 Evaluation Datasets

Evaluation was performed on two publicly available credential benchmark datasets and one independently constructed synthetic evaluation set. The public datasets provide labeled credential samples extracted from real production repositories and real-world secret commits respectively, covering a broad range of credential categories and ground-truth quality levels.

Full dataset identities are available upon request for enterprise due diligence and independent result verification.

A.3 Metrics Definitions

All metrics are computed on binary classification output (secret / not-secret):

- F1: harmonic mean of precision and recall. Primary metric for comparison across configurations.
- Precision: fraction of flagged values that are true secrets. Measures false alert rate.
- Recall: fraction of true secrets that are detected. Measures miss rate.
- MCC (Matthews Correlation Coefficient): balanced metric accounting for all four confusion matrix cells. Particularly informative for class-imbalanced datasets.
- Accuracy: fraction of all samples correctly classified.

Appendix B — Synthetic Evaluation Set Design

The synthetic evaluation set was constructed after training completion to test generalization to credential patterns and code contexts not present in the training distribution. The generator is available for independent verification.

B.1 Design Principles

- Each secret type is paired with realistic code templates drawn from the deployment context where that credential type is commonly found (environment files, Spring Boot properties, Python API clients, shell scripts)
- False-positive types use variable names that reflect genuine non-secret usage (request_id for UUIDs, checksum for SHA-256 digests, nonce for random hex) rather than ambiguous names that would unfairly inflate false-positive rates
- 50 samples per category ensure statistically meaningful per-type recall and FP rate estimates
- No samples were generated using templates that appeared in the training pipeline

B.2 Category Coverage

The synthetic set covers 28 secret categories and 12 false-positive categories, for a total of 1,450 secret samples and 600 non-secret samples (2,050 total). Secret categories include all major cloud provider keys, AI API keys, source control tokens, payment platform keys, communication API credentials, infrastructure secrets, database connection strings, and cryptographic private keys. False-positive categories include all major classes of high-entropy non-secret values observed in production codebases.

B.3 Limitations of Synthetic Evaluation

Synthetic evaluation measures performance on well-formed credentials in structured code contexts. Real production code exhibits greater diversity in variable naming, formatting, encoding, and surrounding context than synthetic templates can reproduce. Reported synthetic metrics should therefore be interpreted as indicative of structured-format detection capability rather than a direct estimate of production recall.