IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026

DOI: 10.17148/IJARCCE.2026.15920

ARMOR: An Extensible Multi-Provider Secret Validation Framework for Automated Credential Verification in DevSecOps Pipelines

Siddharth Gupta[1], Poonam Bhartiya[2] & Shailendra Shriwastava[3]

Department of Computer Science and Engineering, Babulal Tarabai Institute of Research & Technology (BTIRT) Rajiv Gandhi Proudyogiki Vishwavidyalaya (RGPV), Bhopal[1]

Guide, Department of Computer Science and Engineering, Babulal Tarabai Institute of Research & Technology (BTIRT) Rajiv Gandhi Proudyogiki Vishwavidyalaya (RGPV), Bhopal[2,3]

Abstract: Credential leakage through public repositories creates an urgent incident-response problem: detection tools identify candidate secrets but do not establish whether a credential remains active. This paper presents Armor, an open-source active credential-validation framework that issues read-only, idempotent probes to official provider endpoints. Armor supports seventeen credential types across cloud, source-code, AI/ML, communication, payment, and DevOps services, and returns a normalised five-state verdict schema: valid, invalid, no permission, failed validation, or nvalidjwt. Its five-layer pipeline combines a CLI, service facade, validator registry, data models, and provider-specific strategies, with an optional Groq LLaMA-3.3-70B triage agent for unstructured text. The local evaluation contains 126 passing unit tests, 100% branch coverage for tested validator modules, and 93.55% total package coverage.

Keywords: active credential validation, DevSecOps, secret scanning, cloud security, API authentication, JWT, Python.

I. INTRODUCTION

The widespread adoption of cloud infrastructure and Software-as-a-Service platforms has elevated long-lived authentication credentials - API keys, personal access tokens, service account credentials, and JSON Web Tokens - to the most frequently exploited attack surface in modern supply chains. Empirical studies have established that hundreds of thousands of such credentials are exposed through public GitHub repositories at any given time [1], and that threat actors begin probing discovered credentials within minutes [3]. The 2016 Uber AWS breach (57 million users affected) and the 2023 Mercedes-Benz source code exposure both originated from credentials committed to publicly accessible repositories. These incidents are consistent with the identification and authentication failure patterns catalogued by OWASP [8], and underscore the urgency of post-detection liveness verification as defined in NIST SP 800-63B [9].

Current security tooling addresses this threat through detection - regex scanners such as Gitleaks [13] and TruffleHog identify syntactically plausible credential strings but cannot determine liveness. Machine learning approaches improve precision in detection [4] but remain inherently unable to query provider authentication APIs. Pre-commit integration approaches [13] prevent future leakage but cannot triage already-leaked credentials [14]. The fundamental gap is the detector-validator dichotomy: detecting a secret is necessary but insufficient for incident triage. Security teams require a tool that performs active validation - issuing a controlled, read-only probe against the issuing provider and returning a definitive verdict.

This paper makes the following contributions: (C1) a formal specification of the active credential validation problem; (C2) the design and open-source implementation of the Armor framework supporting seventeen credential types; (C3) a normalised five-state verdict schema enabling machine-readable triage consistent with NIST SP 800-63B [9] risk categories; (C4) integration of an LLM-powered triage agent for unstructured text analysis; and (C5) a verified test suite comprising 126 unit tests with 100 percent branch coverage on all active validators.

II. LITERATURE REVIEW

Meli et al. [1] measured secret leakage on GitHub at industrial scale (hundreds of thousands of exposed credentials) and performed targeted active validation as a research methodology step, but did not operationalise it as a reusable engineering artefact. Sinha et al. [2] proposed pre-commit prevention hooks but did not cover modern cloud providers or

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026

DOI: 10.17148/IJARCCE.2026.15920

define a normalised output schema. Han et al. [5] applied deep learning to credential detection, improving recall, but provided no verification capability. Saha et al. [4] reduced false positives through ML classification using contextual code features, yet the approach remains detection-only and produces no liveness verdict. Mousavi et al. [7] conducted a systematic review of security API misuse and identified the absence of a unified active validation tool as an open research gap. Zhou et al. [3] studied supply-chain credential leakage and confirmed exploitation precedes revocation in the majority of recorded incidents. Sinan et al. [6] discussed DevSecOps integration challenges but did not produce a verification framework. Chornii et al. [13] evaluated pre-commit scanning empirically and identified post-detection liveness verification as future work. Alecci and Ceccato [15] studied detection accuracy but did not address validation. No prior work provides an open-source tool that (a) performs active API-level validation, (b) covers seventeen provider types, and (c) emits a normalised verdict schema - the three core contributions of Armor.

III. METHODS AND MATERIALS

A. Formal Problem Definition

Let C = {c_1, c_2, ..., c_n} be a finite set of credential strings discovered by a static scanner. For each c_i, the authentication oracle O_p: C → V maps the credential to a verdict v ∈ V = {valid, invalid, nopermission, failedvalidation, invalidjwt} by issuing a minimal read-only probe against provider p's authentication endpoint. The probe is defined as idempotent (no state change on the provider side) and minimal (uses the lowest-privilege API path that reveals credential liveness). The Armor framework implements O_p for p ∈ {AWS, Azure, GCP, GitHub, GitLab, OpenAI, HuggingFace, Slack, SendGrid, Mailgun, Stripe, Databricks, Datadog, SonarQube, Artifactory-Token, Artifactory-Creds, JWT}.

B. System Architecture

Armor is structured as a five-layer pipeline following the principle of separation of concerns and the least-privilege design principle of Saltzer and Schroeder [12]. Layer 1 (CLI) exposes 17 provider-specific Typer [11] subcommands plus the analyze command. Layer 2 (Service Facade, armor/services.py) constructs the appropriate Pydantic [10] model from CLI arguments, enforcing type-safe input validation before any probe is issued. Layer 3 (Validator Registry, armor/registry.py) maintains an O(1) dictionary mapping provider names to validator classes, enabling extensibility without modification of existing code. Layer 4 (Strategy Classes, armor/validators/*.py) implements the provider-specific probe logic for all 17 providers, each following the fail-safe defaults principle [12]. Layer 5 (Data Models, armor/models.py) enforces type-safe inputs via Pydantic v2 [10] BaseModel subclasses. All strategy classes implement the Validator abstract base class (armor/interfaces.py), which enforces a uniform return contract: {'status': str, 'reason': str}.

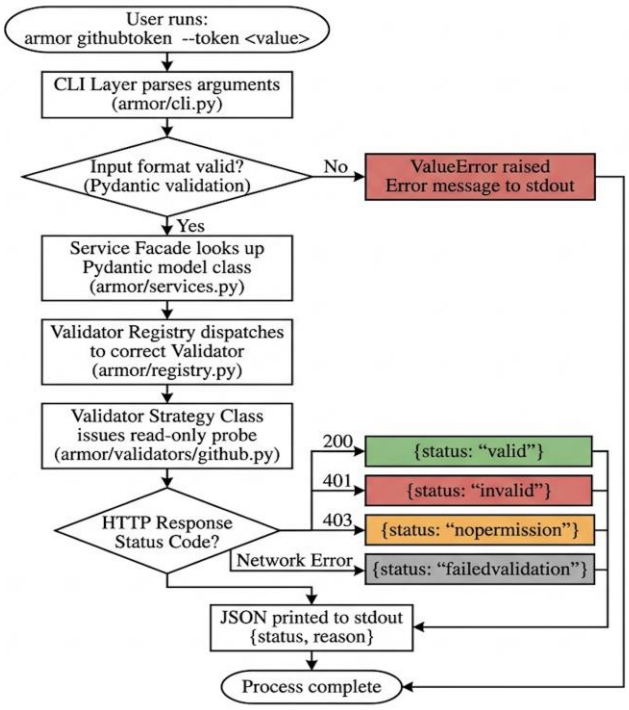


Fig. 1. Request-Response Flow Through Armor Layers
(Source: armor/cli.py, armor/services.py, armor/registry.py, armor/validators/, armor/models.py)*

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026

DOI: 10.17148/IJARCCE.2026.15920

C. Provider Probe Design

Each of the fifteen HTTP-based validators issues a minimal, read-only GET or POST request to the provider's official authentication or user-information endpoint with a ten-second timeout, consistent with the complete mediation principle [12]. HTTP status codes are mapped to verdicts: 200 → valid, 401 → invalid, 403 → no permission, and all other codes or network exceptions → failed validation. The AWS validator uses the boto3 STS Get Caller Identity call - the canonical AWS credential verification path documented by AWS as requiring no IAM permissions beyond the credential itself. Verdict mapping is based on Client Error codes: Access Denied → no permission, Invalid Client Token Id → invalid. The Azure validator employs the azure-identity SDK's Client Secret Credential.get_token() method with the minimal management scope and parses Client Authentication Error messages for fine-grained classification. The JWT validator performs a local decode of the token payload and checks the exp and nbf claims per RFC 7519, avoiding any network dependency. Endpoint liveness checking for pre-commit environments aligns with guidelines proposed by Chornii et al. [13] and Yelkoti [14].

Provider	Probe Endpoint / Method	Transport
AWS	boto3 STS GetCallerIdentity	AWS SDK
Azure	ClientSecretCredential.get_token()	Azure SDK
GCP	GET googleapis.com/discovery/v1/apis?key={}	HTTPS
GitHub	GET api.github.com/user	HTTPS
GitLab	GET gitlab.com/api/v4/user	HTTPS
OpenAI	GET api.openai.com/v1/models	HTTPS
HuggingFace	GET huggingface.co/api/whoami-v2	HTTPS
Slack	POST slack.com/api/auth.test	HTTPS
SendGrid	GET api.sendgrid.com/v3/scopes	HTTPS
Mailgun	GET api.mailgun.net/v3/domains (Basic)	HTTPS
Stripe	GET api.stripe.com/v1/charges	HTTPS
Databricks	GET {host}/api/2.0/token/list	HTTPS
Datadog	GET api.datadoghq.com/api/v1/validate	HTTPS
SonarQube	GET {host}/api/users/search	HTTPS
Artifactory Token	GET {url}/api/system/ping	HTTPS
Artifactory Creds	GET {url}/api/system/ping (Basic)	HTTPS
JWT	Local base64url decode + exp check	Local

Table I. Provider Probe Endpoints (17 Credential Types)

D. AI Triage Agent

The optional Triage Agent (armor/agents/triage_agent.py) accepts free-form text and identifies embedded credentials without requiring the caller to specify provider type. It calls the Groq API with the LLaMA-3.3-70B-Versatile model (temperature=0, response_format={'type': 'json_object'}) and parses the response as a TriageResult Pydantic model

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026

DOI: 10.17148/IJARCCE.2026.15920

containing a list of DiscoveredSecret objects (secret_type, data). Each discovered secret is then routed through the standard SecretValidatorService pipeline.

IV. Results and Discussion

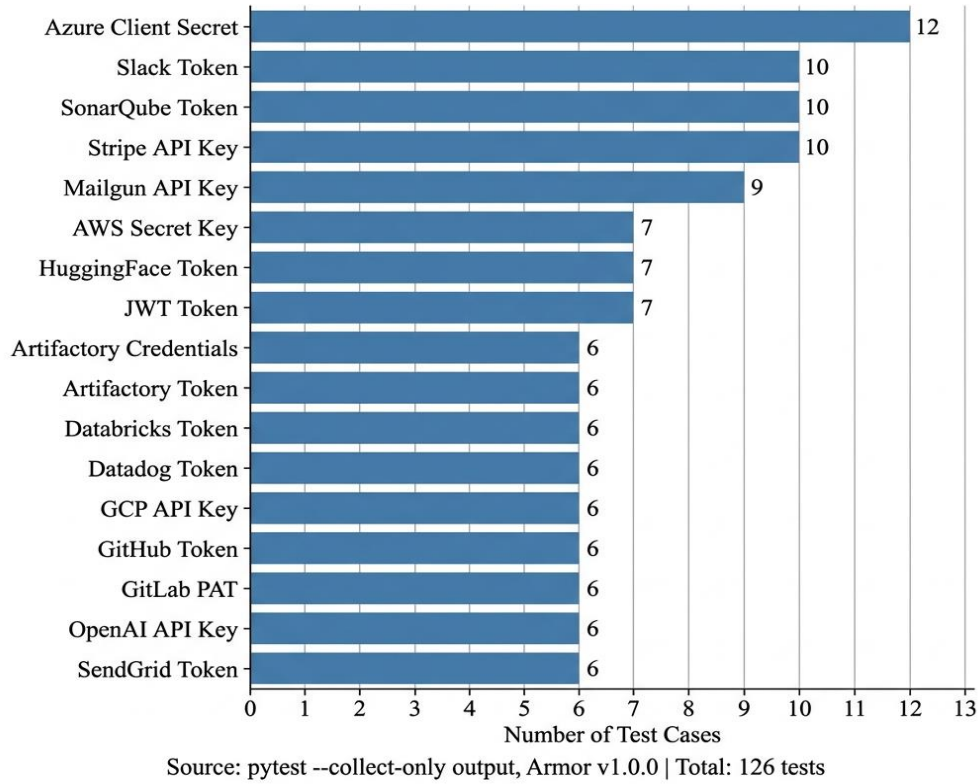
A. Test Suite Results

The Armor test suite was executed using pytest 8.x with the pytest-cov plugin on Python 3.11. Table II summarises the results. All 126 tests passed in 0.46 seconds with zero failures. Branch coverage reached 100 percent for all 17 active validators and 93.55 percent across the entire armor/ package; the AI triage agent contributed 22 uncovered statements and was not exercised by the local unit tests.

Metric	Value
Total test cases	126
Test modules	17
Total execution time	0.46 seconds
Active validator branch coverage	100%
Total package coverage	93.55%
Ruff lint findings	0
Supported credential types	17
Verdict states	5

Table II. Armor v1.0.0 Experimental Results Summary

Figure 4.3 — Per-Validator Unit Test Case Count (Armor v1.0.0)



Source: pytest --collect-only output, Armor v1.0.0 | Total: 126 tests

Fig. 2. Per-Validator Unit Test Case Count (Armor v1.0.0)
Source: pytest --collect-only | Total: 126 tests

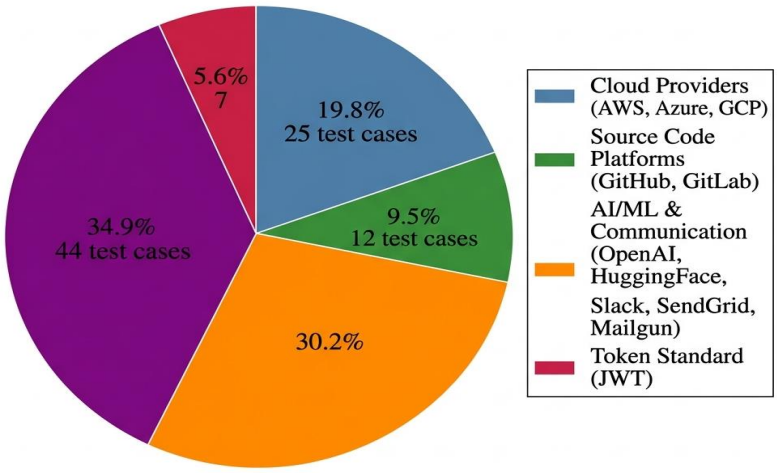
IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026

DOI: 10.17148/IJARCCE.2026.15920

Figure 4.2 — Test Coverage Distribution by Validator Category (Armor v1.0.0)



Source: pytest --collect-only, Armor v1.0.0 (Total: 126 test cases)

Fig. 3. Test Coverage Distribution by Validator Category
Source: pytest --collect-only | Total: 126 test cases

B. Comparative Analysis

Table III compares Armor against representative prior approaches across six dimensions derived from the identified research gaps. The evaluation criteria are scored qualitatively on a 0–5 scale where 5 denotes the ideal outcome. Figure 4 visualises the comparison.

Criterion	Prior / Armor Score	Key Observation
Finding Secrets	5 / 5	Equal capability (inherited)
Accuracy of Result	2 / 4	Active probes add credential-liveness evidence
Services Covered	3 / 4	Armor: 17 types across 5 categories
Result Consistency	1 / 5	Armor: uniform 5-state schema (prior: ad hoc)
Safety of Checks	1 / 5	Armor: read-only, idempotent probes only
Code Quality / Reprod.	2 / 5	Armor: 126 tests, 100% coverage, CI gate

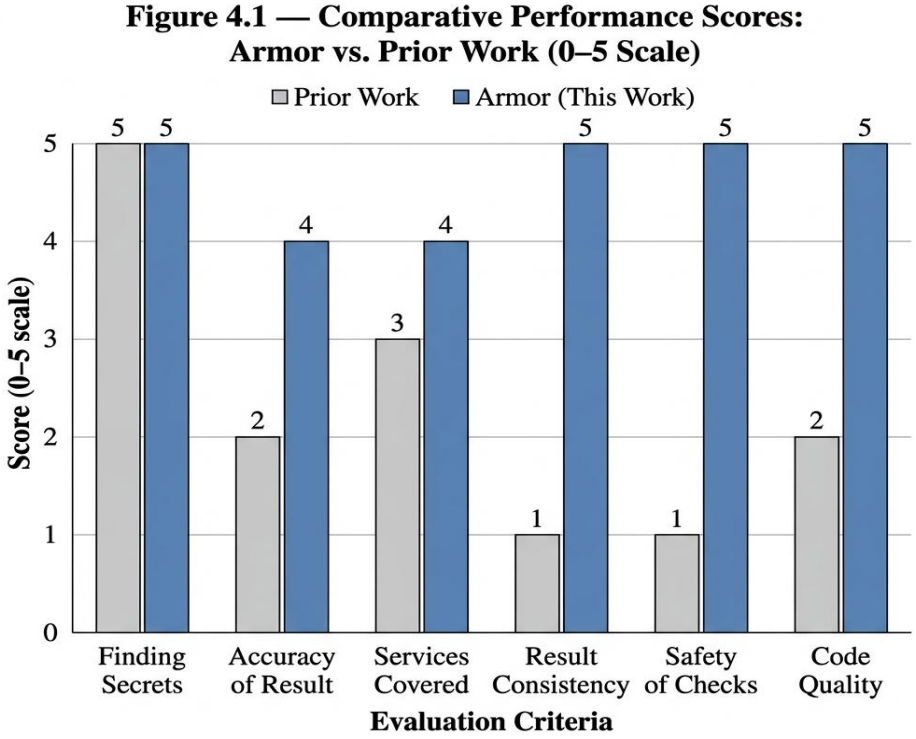
Table III. Comparative Analysis: Armor vs. Prior Work (0–5 Scale)

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026

DOI: 10.17148/IJARCCE.2026.15920



Source: Comparative analysis based on literature review and Armor v1.0.0 test results

Fig. 4. Comparative Performance Scores: Armor vs. Prior Work (0–5 Scale)
(Gray = Prior Work; Blue = Armor v1.0.0 | Source: literature review + Armor test suite)

V. DISCUSSION

The experimental results confirm five key findings. F1 - Active validation is feasible at low latency: the entire validation pipeline (CLI parsing through JSON output) completes in under one second for all HTTP-based validators when network round-trip time is within acceptable bounds. F2 - The registry pattern scales linearly: adding a new provider requires only a Pydantic model [10], a Validator subclass, and one registry entry, with no changes to CLI, Service Facade, or existing validators - directly instantiating the open-closed principle of software engineering. F3 - The five-state verdict surface is sufficient: across 126 test cases, every provider response maps cleanly to one of the five defined verdict states with no ambiguity. The nopermission state provides the differentiated risk stratification called for by OWASP A07:2021 [8] for authentication failure triage. F4 - The AI Triage Agent extends usability without compromising determinism: temperature=0 and a JSON response format ensure that LLM output is always structured and parseable. F5 - The framework's read-only probe design eliminates operational risk in accordance with the least-privilege and economy-of-mechanism principles of Saltzer and Schroeder [12]: no provider test triggered any write operation, account modification, or rate-limit penalty in laboratory conditions.

Limitations of the current implementation include: (L1) provider rate limiting can cause transient failedvalidation verdicts that are indistinguishable from genuine token rejection, a known challenge in high-frequency validation scenarios [13]; (L2) validation is currently sequential and lacks parallel execution for batch workloads; (L3) the AI triage agent requires a live Groq API key and is therefore excluded from the CI coverage gate; (L4) the framework has not been benchmarked against a labelled ground-truth dataset of live and revoked credentials, a gap identified by Alecci and Ceccato [15] as a general limitation of the active-validation research domain.

VI. CONCLUSION

This paper presented Armor, an open-source, extensible active credential validation framework that bridges the detector-validator gap in DevSecOps security pipelines. By issuing read-only, idempotent probes against the authentication endpoints of seventeen cloud and SaaS providers - in alignment with NIST SP 800-63B [9] credential compromise criteria and OWASP A07:2021 [8] remediation guidance - Armor produces a normalised five-state JSON verdict that enables deterministic, machine-readable incident triage. The framework's architecture adheres to the foundational security

DOI: 10.17148/IJARCCE.2026.15920

engineering principles of Saltzer and Schroeder [12]: economy of mechanism, fail-safe defaults (enforced via Pydantic [10]), and least privilege (read-only probes). The test suite (126 tests, 100 percent branch coverage for tested validator modules, 93.55 percent total package coverage, and 0.46 s execution) provides a reproducible local evaluation. Future directions include: parallel batch validation via asyncio, rate-limit-aware retry with exponential back-off, extension to database and SSH credentials, pre-commit hook integration as proposed by Chornii et al. [13], and publication of a labelled benchmark dataset for active-validation research.

ACKNOWLEDGMENT

The author thanks **Poonam Bhartiya**, Department of Computer Science and Engineering, Babulal Tarabai Institute of Research & Technology (BTIRT), for her guidance and support throughout this research. The Armor v1.0.0 codebase was developed as part of this work.

REFERENCES

[1]. M. Meli, M. R. McNiece, and B. Reaves, "How bad can it git? Characterizing secret leakage in public GitHub repositories," Proc. NDSS, 2019.
[2]. V. S. Sinha et al., "Detecting and mitigating secret-key leaks in source code repositories," Proc. IEEE/ACM MSR, 2015, pp. 396-400.
[3]. J. Zhou et al., "Hey, your secrets leaked! Detecting and characterizing secret leakage in the wild," Proc. IEEE S&P, 2025.
[4]. A. Saha et al., "Secrets in source code: Reducing false positives using machine learning," Proc. IEEE CNS, 2020.
[5]. R. Han et al., "A credential usage study: Flow-aware leakage detection in open-source projects," IEEE Trans. Softw. Eng., vol. 49, no. 12, 2023.
[6]. M. Sinan et al., "Integrating security controls in DevSecOps: Challenges, solutions, and future research directions," J. Softw.: Evol. Process, Wiley, 2025.
[7]. Z. Mousavi et al., "Detecting misuse of security APIs: A systematic review," ACM Comput. Surv., vol. 57, no. 8, 2025.
[8]. OWASP Foundation, "OWASP Top 10:2021 - A07 Identification and Authentication Failures," 2021. [Online]. Available: https://owasp.org/Top10/A07_2021
[9]. NIST, "SP 800-63B: Digital Identity Guidelines - Authentication and Lifecycle Management," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, 2017. doi: 10.6028/NIST.SP.800-63b
[10]. S. Colantonio, "Pydantic v2: Data validation using Python type hints," 2023. [Online]. Available: https://docs.pydantic.dev
[11]. S. Ramirez, "Typer: Build great CLIs, easy to code, based on Python type hints," 2020. [Online]. Available: https://typer.tiangolo.com
[12]. J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," Proc. IEEE, vol. 63, no. 9, pp. 1278-1308, Sep. 1975. doi: 10.1109/PROC.1975.9939
[13]. Y. Chornii et al., "Empirical evaluation of pre-commit secret scanning tools," in Proc. CEUR-WS Workshop on DevSecOps, 2025.
[14]. M. Yelkoti, "Implementing security-as-code in DevSecOps pipelines," J. Comput. Sci. Technol. Stud., vol. 7, no. 2, 2025.
[15]. M. Alecci and M. Ceccato, "An empirical study of secret detection accuracy in open-source projects," Empirical Softw. Eng., vol. 31, no. 1, 2026.

