Sri Lanka Institute of Information Technology

Information Warfare(IW) – 2025

Design and Development of a Commercialisable Product

Project

LeakScanner – A Multi-Tenant Configuration-Based Source Code Leak Detection Tool

IT22356222

K.B.H.M.Thisiru Tharupathi Bandaranayake

Table of Contents

ABSTRACT

In both public and private Git repositories, hard-coded secrets and proprietary identifiers commonly leak, giving attackers access to database credentials, API keys, and internal system information. Source-code leaks are a serious information warfare and cybersecurity concern because, according to extensive empirical research, thousands of secrets are leaked on GitHub every day, and many of them stay public for extended periods of time [1]. Current secret-scanning tools, like GitGuardian, Gitleaks, and related DevSecOps integrations, are typically provided as SaaS platforms and primarily target individual organizations, making them unsuitable for all threat models or regulatory environments [2].

In order to search GitHub repositories for organization-specific code and credential exposure, this paper introduces LeakScanner, a configuration-driven, multi-tenant leak detection tool. LeakScanner allows a single deployment to safely serve numerous organizations by separating tenant configuration (brand keywords, domains, regex patterns, and scan policies) from the core engine. To determine the probability that a finding actually belongs to a particular organization, the tool integrates pattern-based detection, GitHub's code search API, and optional large language model (LLM) analysis through Google Gemini. The prototype includes secure environment-based credential handling, per-tenant reports in Markdown and JSON, and a test mode that creates realistic mock findings for demonstrations without interacting with live repositories. The work shows how a local-first, relatively lightweight architecture can approximate some of the features of commercial secret-scanning platforms while still being adaptable, extensible, and appropriate for use in research and education related to information warfare.

INTRODUCTION

Workflows for software development are depending more and more on hosting platforms like GitHub and GitLab as well as distributed version control systems like Git. These platforms facilitate collaboration, but they also open up new avenues for attack when developers inadvertently upload private data to source code repositories. Previous extensive measurements have demonstrated that passwords, cryptographic keys, and API keys are frequently leaked into public repositories and that many of these secrets continue to be valid for a long time after being made public. [1] Significant amounts of credential leakage in open-source and educational projects have been reported in more recent research, confirming that this is a recurring and systemic issue. [3]

Hard-coded secrets in code bases compromise confidentiality, authentication, and permission, allowing hackers to switch between cloud infrastructures, impersonate services, and steal data. [4]In addition to explicit credentials, companies are vulnerable to the disclosure of proprietary identifiers, such as internal project names, system codenames, and internal email patterns, which can facilitate targeted phishing and intelligence collection or divulge key architecture information. Both academia and business have suggested automated methods for

secret detection to reduce these dangers. These methods include machine learning, regular expression-based scanning, and, more recently, LLM-based analysis. [5]

To stop secrets from being pushed to shared repositories, DevSecOps pipelines frequently use commercial technologies like GitGuardian, Gitleaks, and other secret-scanning services. [2]

Nevertheless, a lot of these solutions are offered as SaaS platforms that are focused on the environment and regulations of a single company. Local-first systems that may be set up on-site and customized for each client without centralizing telemetry in a third-party cloud are preferred for certain security-sensitive or regulated businesses. Simultaneously, multi-tenant tooling a single engine that can scan on behalf of several client businesses, each with its own brand, domains, and detection rules is becoming more and more necessary for security service providers. LeakScanner, a dynamic configuration-based leak detection tool that supports numerous tenants via JSON configuration files, is designed and implemented in this work to close this gap. While the fundamental engine stays the same, each tenant defines its own brand identities, domains, keywords, and regex patterns. The scanner performs configurable pattern-based detection, retrieves potential files using GitHub's code search API, and optionally employs a big language model (Google Gemini) to determine whether a code sample is likely to belong to the configured organization.

The prototype's two main objectives are to:

(1) provide a workable, expandable prototype that can be commercialized and patented in the future;
(2) serve as a teaching tool for information-warfare courses by illustrating the actual dangers of source-code leaks and defensive scanning techniques.

RELATED WORK

Secret Leakage in Source Code Repositories

The frequency and effects of secret leaks in public repositories have been measured in a number of empirical investigations. One of the first extensive examinations of GitHub was carried out by Meli et al., who demonstrated that thousands of new secrets are added every day and that secrets like API keys and cryptographic content are regularly committed. [4]

PassFinder, an automated method for identifying password leaks on GitHub, showed that numerous hard-coded passwords exist in a variety of languages and projects. [1] Even when developers are aware of the hazards, inadvertent exposure is nonetheless widespread, according to more recent research that looked at credential leakage in certain contexts, including undergraduate open-source projects or specialized ecosystems. [3]

Mixed-methods research has investigated how developers manage secrets in practice, going beyond merely quantitative assessments. Time constraints, a lack of tools, and ambiguous accountability boundaries frequently result in unsafe practices, according to Krause et al.'s study of processes and mental models around the storage and management of secrets in

repositories. [6] Together, these findings highlight the need for more efficient, integrated detection tools as well as improved developer education.

Secret-Scanning Tools and DevSecOps Integration

The industry has developed an expanding ecosystem of secret-scanning tools in response to these threats. An enterprise-grade secrets detection engine offered by GitGuardian searches repositories and DevOps tools for a variety of credential types, such as database credentials, API keys, and multi-line secrets like private keys. [2] With connections with CI/CD pipelines and pre-commit hooks, Gitleaks provides an open-source substitute that focuses on regex-based detection across Git repositories. The goal of other programs like TruffleHog, Yelp's Detect-Secrets, and platform-native capabilities like GitHub Advanced Security secret scanning is to identify hard-coded credentials before they are used in production settings. [6]

To lower false positives, these technologies usually rely on carefully chosen rule sets, pattern-matching heuristics, and in certain situations machine learning models. They are good at identifying common credential formats (such AWS access keys and OAuth tokens), and many have backlog triage, dashboards, and alerting features that are appropriate for big businesses. However, the majority of commercial products function as hosted SaaS services and are built around a secure perimeter for a single enterprise. In situations where a security provider must scan code for several independent clients or where stringent data-sovereignty requirements prohibit sending code metadata to third-party clouds, they may not be appropriate because they frequently assume control over the code hosting platform or deep integration into that organization's CI/CD system. [5]

Advanced Techniques and AI-Assisted Secret Detection

Machine learning and language model-based methods for secret detection have been investigated recently. In order to differentiate real secrets from harmless lookalikes in source code, Konygin presented a bigram-based model, emphasizing that simple regular expressions by themselves can result in large false-positive rates. [5]

Semantic analysis and machine learning classifiers have been used in other research to improve context understanding and lower noise in detection pipelines. Large language models (LLMs) have emerged, and recent research has started to analyze LLMs for detecting secret-like patterns and evaluating leakage risks in code, claiming that they can capture more contextual information than set regex rules. [7] [8]

However, other writers warn that in order for secret detection technologies to be effective, they must be used in conjunction with organizational procedures, remediation workflows, and secure development standards. At the nexus of these developments, LeakScanner maintains visible, adjustable pattern-based scanning while potentially assigning higher-level judgments (such as "**does this snippet look like it belongs to Customer X?**") to an LLM. This architecture enables AI-assisted triage when necessary while maintaining the fundamental detection logic's comprehensibility and auditability. [7]

Gap and Contribution

To the best of our knowledge, very few academic or open-source solutions are specifically made to be multi-tenant, configuration-driven leak scanners that security service providers can deploy locally first. With less emphasis on per-customer branding, pattern customisation, and strict data localization, current technologies either concentrate on single-tenant organizational deployments or function as centralized SaaS services. [2]

LeakScanner offers a lightweight design where the scanning engine is generic and each tenant is defined by a JSON configuration that includes brand keywords, domain indicators, regex patterns, and scan parameters. This facilitates offline demonstrations through test mode, streamlines the onboarding process for new clients, and offers a clear route to commercialization as a multi-tenant defensive information-warfare weapon. The system architecture, implementation, market positioning, and ethical issues of this design are described in detail in later sections of the paper.

SYSTEM ARCHITECTURE DESIGN

LeakScanner is intended to be a multi-tenant, configuration-driven leak detection system. The main idea is to clearly distinguish between the scanning process (the engine) and the content being scanned (tenant-specific metadata and patterns). This makes it possible to securely serve several enterprises with a single deployment without changing the code for each client.

Design Goals

The following objectives served as the foundation for the architecture:

1. Multi-tenancy: Without altering the core code, support numerous companies (tenants) with unique branding, domains, and secret patterns.
2. Configuration-driven behavior: JSON configuration files should be used to specify all tenant-specific behavior.
3. Local-first and privacy-conscious: Permit the scanner to operate locally without requiring data to pass via a third-party SaaS backend.
4. Extensibility: Make it simple to incorporate new source platforms and detection rules (regex patterns, keywords, AI analysis) in the future.
5. Explainability: To enable auditing and justification of findings, keep detection logic (patterns + configuration) transparent.

High Level Architecture

The system is conceptually divided into five major parts:

1. Configuration Layer: Per-tenant JSON files located in configs/ that provide scan parameters, domains, brand keywords, regex patterns, and variable names for API credentials.
2. Scanning Engine: A Python CLI program (leakscanner.py) that coordinates configuration loading, external provider queries (like GitHub), pattern matching, and findings aggregation.
3. A collection of features that communicate with external code hosting systems is known as the source integration layer. The GitHub REST API is used in the present prototype to implement code search and file content retrieval.
4. Analysis Layer: Regular expressions are used for pattern-based detection, and each finding is given a verdict and degree of confidence utilizing optional LLM-based reasoning (Google Gemini).
5. Persistence and Reporting Layer: Log files for auditing and debugging, as well as JSON and Markdown per-tenant result storage.

The system can be expanded without significant connection between layers thanks to these components' communication via well-defined data structures (JSON objects and Python dictionaries).

Configuration Model

There is a configuration file called <customer_id> for every tenant.json can be found in the subfolder configs/. Usually, the configuration model consists of:

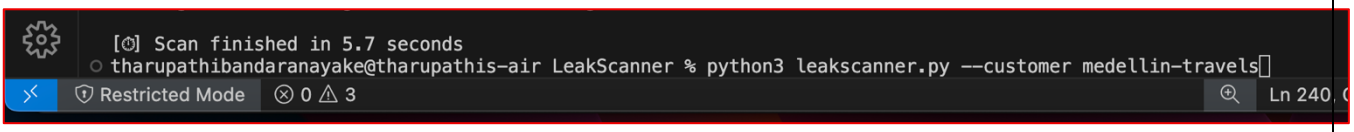
- Customer metadata: Customer_id, customer_name, and a brief description.
- Domain names, email suffixes, internal project names, and code prefixes that define the tenant's identity are examples of brand metadata.
- Keywords: A collection of strings used as the key search phrases in code search queries, such as brand names, internal system names, and domains.
- Regex patterns: A collection of pattern objects, each of which has:
    - name: a descriptive identifier
    - pattern: a consistent expression,
    - description: an explanation of what it finds,
    - severity: critical, high, medium, etc.
    - enabled: flag to change the rule's status.
- Scan parameters include fields like max_results_per_keyword, search_types, exclude_repos, and context_lines (the quantity of surrounding code to capture).
- AI settings (optional): Choose the model to use, whether AI is enabled, and optional tweaking parameters like batch size.

- API credentials mapping: Environment variable names for the Gemini key and GitHub token, so the code doesn't care how secrets are kept.

Data Flow

1. CLI Invocation:

USER RUNS THE COMMAND



2. Configuration Loading: The engine finds **configs/<customer_id>.json**, parses it, and verifies that the necessary fields are there. Through the use of.**env,** environment variables **(such as GITHUB_TOKEN)** are loaded and compared to the names specified in the configuration.
3. Keyword Based Search: Unless it is operating in test mode, the engine sends a code search query to GitHub's API for every keyword in the configuration.
4. File retrieval and context extraction: The raw file content is retrieved for every item that matches. The context lines that surround the keyword occurrence are extracted by the engine.
5. Pattern-based analysis: Zero or more pattern matches with corresponding severities are produced by scanning the file content and context against the preset regex patterns.
6. AI-assisted analysis (optional): The tool transmits a condensed context snippet and metadata (customer name, keyword) to Gemini, which provides a textual verdict and confidence, if AI is enabled and an API key is supplied.
7. Finding aggregation: A finding object including the type, keyword, repository, file location, matched line, severity, AI verdict, and pattern matches is generated for every candidate.
8. Reporting: After the scan is complete, the results are shown in a Markdown report for human review, written to a JSON file for programmatic consumption, and summarized in the console. Errors and specific occurrences are recorded in a log file.

In order to demonstrate the entire process without network connection, a different test mode synthesizes actual findings instead of using external API calls.

IMPLEMENTATION DETAILS

LeakScanner is a single-file CLI utility with a directory of configuration files that is implemented in Python 3. The main modules and implementation options are covered in this section.

Technologies and Dependencies

A small number of dependencies are used in the prototype:

For basic logic, configuration parsing, logging, regex evaluation, and scheduling, Python's standard library includes argparse, json, logging, re, datetime, pathlib, and time.

- **requests:** for interacting with the GitHub REST API via HTTP.
- **python-dotenv:** To load API keys into environment variables from a **.env** file.
- **Google-generativeAI:** To do AI-assisted leak analysis with optional integration with Google Gemini models.

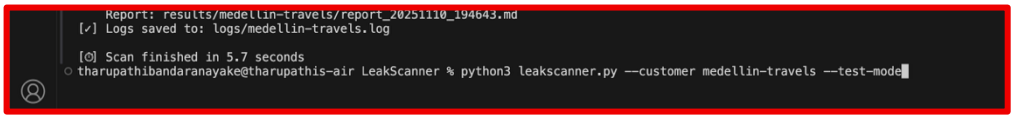
The tool will remain lightweight and simple to use in standard developer or server contexts thanks to this minimum infrastructure.

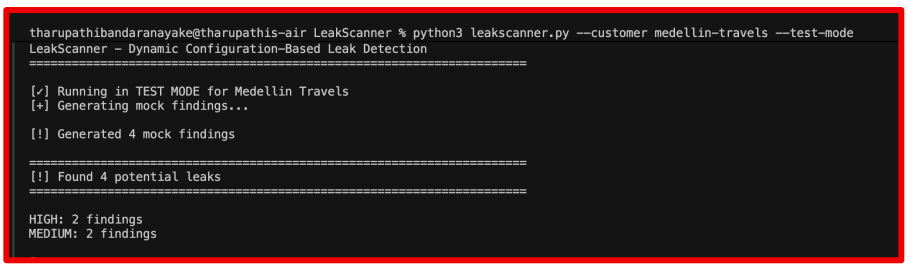
Command-Line Interface

Python's argparse module is used in the construction of the CLI. The primary arguments are:

--customer: The tenant identifier, which must match a JSON file in the configs/ directory.

--test-mode: The program operates in a self-contained simulation mode and produces fictitious results rather than querying GitHub.





--limit: supersedes the configuration file's maximum results per keyword.

--out-json: a custom output path for the JSON results file; if not, results/<customer_id>/. is where a timestamped file is created.

The tool is scriptable and simple to incorporate into batch processes or continuous integration pipelines thanks to its interface.

Configuration Loading and Validation

Using the customer ID, the configuration loader creates a file path and tries to read **configs/<customer_id>.json.** The application gracefully terminates with a clear error notice if the file is missing or contains invalid JSON.

Before continuing, basic validation makes sure that essential parts (such as **keywords, regex_patterns,** and **scan_settings**) are present. The **api_credentials** section is used to read API credential names, and environment variables are used to retrieve the appropriate values. The actual tokens are kept outside of version control by loading values from a local .**env** file at startup using the **python-dotenv** module.

GitHub API Integration

Two functions are used to implement the integration with GitHub:

1. (**keyword, github_token, repo_filter, limit**) search_github_code
    a. Uses the GitHub code search API endpoint to create a search query.
    b. If a token is available, an Authorization header is added.
    c. Accepts a limit on the number of results per keyword and an optional repo_filter.
    d. Returns either None in the event of an error or the parsed JSON answer.
2. **get_file_content** (**file_path, github_token, repo_full_name**)
    a. Retrieves the raw file contents by querying the contents endpoint.
    b. If successful, the file is returned as text; if unsuccessful, it is returned as None.

Basic error handling and logging for network outages and HTTP issues are included in the implementation. To lessen the chance of exceeding rate constraints, a brief pause is included in between requests. Explicit rate-limit inspection and backoff techniques could improve this in a production version.

Pattern Matching and Context Extraction

The engine conducts two primary analysis for every file that is retrieved:

1. Context extraction involves dividing the file into lines and identifying the lines that contain the term (case-insensitive match). A context window (e.g., 100 lines before and after) is extracted for every match using **context_lines** from the configuration. Both human assessment and AI analysis can benefit from this background.

2. Regex evaluation: Every enabled regex pattern specified in the settings is compared to the entire file content. To discover every match, using Python's re.finditer. The code logs the following for every match:
    - pattern name,
    - description,
    - severity,
    - a condensed version of the text that matched,
    - the match's character position.

AI Assisted Analysis

The AI Analysis Module Is Intended To Be Optional. When A Gemini API Key Is Available And Activated, The Engine:

1. Uses The Supplied API Key To Configure The Google-Generativeai Client.
2. Creates A Gemini Model (Such As Gemini-1.5-Flash).
3. Sends A Prompt With The Following Information:
    - The Name Of The Client,
    - The Term,
    - A Condensed Piece Of Code (The Match's Surrounding Context),
    - Directives To Reply With A Structured Judgment (YES/NO/UNCERTAIN) And Level Of Confidence (High/Medium/Low).

Simple Fields Are Extracted By Parsing The Answer Text:

- Verdict HAS {Yes, No, Uncertain, Error},
- Confidence HAS {High, Medium, Low, N/A}.

These fields can be used to prioritize triage and are kept in the finding object. Findings simply record a placeholder judgment (such as **not_analyzed**) while maintaining pattern-based severity if AI is disabled or unavailable.

Test Mode and Mock Data

LeakScanner has a test mode to enable offline demonstrations, quick testing, and secure assessment without requiring external API requirements. When the flag --test-mode is enabled:

- All calls to GitHub are avoided by the tool.
- For the chosen tenant, a small, deterministic collection of mock results is produced, imitating situations like:
    - An exposed.env file with an API key in it.
    - An internal system name is referenced in a configuration file.
- Logs are written and JSON and Markdown reports are generated using the same reporting pipeline.

In situations involving teaching and presentations, when network connectivity or rate limitations could be erratic, this mode is especially helpful.

Reporting and Logging

The engine generates a tenant-specific directory under **results/<customer_id>/** for every scan and writes:

- A JSON file that includes:
  - scan the timestamp,
  - client ID,
  - total number of discoveries,
  - a list of items to be found.
- A Markdown report that summarizes:
  - basic metadata (date, customer, total discoveries)
  - a section with the kind, repository, file location, keyword, severity, URL, and, if provided, the AI verdict for each finding.

Python's logging package is used to write logs to **logs/<customer_id>.log**. Start and end times, keywords scanned, errors found, and summary data are all recorded in the logs. Debugging, auditing, and possible future integration with monitoring tools are all supported by this.

MARKET AND COMPETITOR ANALYSIS

The attack surface associated with source-code repositories has been greatly increased by the growing reliance on cloud-native architectures, continuous integration/continuous deployment (CI/CD) pipelines, and distributed software teams. According to public studies, thousands of new hard-coded secrets appear on platforms like GitHub every day, and many of these secrets remain valid long after initial exposure. Misconfigured repositories and accidental credential commits have been identified as key factors in several high-profile supply-chain attacks. As a result, secret detection and repository-security tooling has become a separate product category within the larger application security (AppSec) and DevSecOps markets.

This area is the focus of several open-source and commercial tools. GitGuardian provides SaaS dashboards, compliance reporting, and remedial workflows while concentrating on large-scale secret detection across repositories, registries, and CI systems. Gitleaks and TruffleHog are open-source command-line programs designed to be integrated into CI/CD pipelines and pre-commit hooks that search Git history for recognized secret patterns. By offering a customized pre-commit architecture, Yelp's Detect-Secrets project similarly seeks to stop credentials from entering repositories. Additionally, platform providers have included native tools that incorporate policy enforcement and secret detection right into their

ecosystems, such Microsoft Defender for DevOps and GitHub Advanced Security secret scanning.

Although these solutions work well in many business settings, they have a number of drawbacks when compared to the issue that LeakScanner attempts to solve. First, the majority of solutions are made for single-tenant installations, meaning that the security perimeter and repositories are assumed to be owned by a single company. Custom scripting, independent deployments, or manual policy management are frequently needed for multi-tenant use, in which a managed security company or consultant scans on behalf of multiple independent clients. Second, a lot of top products are largely offered as SaaS platforms, which could go against client desires for on-premises, local-first tooling or regulatory restrictions. Lastly, it may be more difficult to match rules with internal system names, organization-specific branding, or specialized credential formats if proprietary engines offer little insight into detection algorithms.

LeakScanner is in a position to fill in these gaps. The core of its value proposition is:

- **Multi-tenant configuration:** One engine can serve numerous enterprises while maintaining logical separation of branding, patterns, and thresholds because each client is represented by a different JSON configuration.
- **Local-first operation:** The code hosting platform (such as GitHub) is the only external dependence; all scanning, pattern analysis, and report production take place on the operator's computer.
- **Hybrid detection:** Deterministic detection of known secret formats and contextual evaluation of organization-specific leaks are made possible by a combination of transparent regex-based rules and optional large language model (LLM) verdicts.
- **Audit-friendly outputs:** Without the need for proprietary dashboards, plain JSON and Markdown reports facilitate integration into ticketing systems, compliance evidence, and client deliverables.

As businesses adopt DevSecOps methods and regulators place an emphasis on internal control of sensitive data, the global market for application security and secret-management solutions is expected to rise significantly. LeakScanner targets managed security service providers (MSSPs), small-to-medium-sized businesses (SMEs) that need inexpensive local scanning, and academic or research institutions that require safe, repeatable demonstrations of defensive information warfare concepts and source-code leakage.

INTELLECTUAL PROPERTY AND PATENTABILITY ASSESSMENT

Novelty and Inventive Aspects

From an intellectual property standpoint, LeakScanner incorporates a number of design components that, when combined, could create an invention that qualifies for a patent:

Multi-tenant model driven by configuration: Only structured configuration (JSON) is used to describe each tenant, including scan policies, keywords, regex patterns, and brand metadata. Because the engine is unaffected by tenant identification, it can support numerous logically distinct customers in a single deployment.

Context-aware hybrid analysis: To determine whether a code snippet is likely connected to a specific tenant's brand or internal systems, the tool uses pattern-based detection and can optionally call an LLM. Each finding includes the AI's decision, which adds contextual reasoning to static patterns.

Test-mode simulation pipeline: Without requiring access to actual repositories, a special test mode exercises the entire reporting and logging pipeline by creating realistic leak scenarios for any configured tenant. This makes training and demonstration in DevSecOps and information-warfare environments safe.

Strong secret detection and DevSecOps integrations are implemented by current tools like GitGuardian, Gitleaks, and TruffleHog, although publicly accessible documentation and literature indicate that they are primarily intended to be either single-organization solutions or centralized SaaS products. Neither explicit AI-driven ownership evaluation of leaked code snippets nor per-tenant configuration as a first-class architectural element are highlighted by them. This implies that, subject to a formal prior-art investigation, LeakScanner's architecture has a fair probability of satisfying innovation and non-obviousness requirements in numerous jurisdictions.

Potential Claims And Protection Strategy

A prospective patent application might concentrate on system and method claims like:

- A technique for multi-tenant leak detection in code repositories where a single scanning engine performs searches and assesses results based on the active tenant configuration. Each tenant is defined by a configuration that describes brand keywords, patterns, and thresholds.
- A technique that combines rule-based and LLM-based analysis, including encoding the LLM verdict into a structured finding record, to ascertain if a found code segment is associated with a specific organization.
- A system that uses configuration-driven fake discoveries to test detection and reporting algorithms without requiring external dependencies and offers a simulated leak scenario for each tenant.

Standard IP measures could include the following in addition to patent protection:

- **Copyright:** claiming ownership of the configuration schemas, documentation, and source code.
- **Trademarks:** registering LeakScanner and related software and cybersecurity service branding.
- **Licensing:** employing a dual-licensing approach (for example, a commercial license for multi-tenant deployments and a permissive or educational license for the test-mode version).

Limitations and Risks

IP strategy may be limited by a number of factors:

- Certain characteristic combinations may become less unique over time due to the quick development of secret-scanning and AI-assisted security tools.
- varied jurisdictions have varied interpretations of software patents; in some, proving a technical impact beyond generic computer implementation is crucial.
- Due to prior art from established tools and academic prototypes, too general claims that try to cover generic secret scanning run the danger of being rejected.

Despite these restrictions, it would assist maintain freedom of operation and set precedent to record the design in an internal invention disclosure and, if desired, file a provisional patent or defensive publishing.

ETHICAL AND RISK CONSIDERATION

LeakScanner’s design and implementation must take into account a number of ethical and legal issues because it operates at the nexus of cybersecurity, privacy, and information warfare.

Authorized Use and Consent

Potentially sensitive content is inevitably touched upon when searching repositories for compromised credentials or proprietary identifiers. Operators must only scan for ethical use:

- Repositories That They Manage Or Own, Or
- Repositories For Which The Owner Organization Has Given Them Specific Authority.

Even if third-party code is publicly accessible, unauthorized scanning of it may be against local computer usage laws or terms of service, particularly if the results are retained or used offensively. Therefore, any commercialization should include terms that limit users to genuine, defensive use cases in line with responsible disclosure norms, and tools like LeakScanner should have unambiguous usage warnings.

Data Protection and Privacy

LeakScanner's reports may include sensitive metadata, such as repository URLs, configuration file snippets, and indications about internal system names or credentials, even though it is intended to be local-first and does not store raw source code outside of the operator's environment. These results ought to be handled as private artifacts:

- Reports must be kept in designated areas.
- Only authorized security professionals should have access to records and results.
- The amount of time that results are retained before being archived or erased should be specified by retention policies.

If AI analysis is enabled, care must be made to check the provider's data-handling policies and avoid sending more code context to external LLM APIs than is required for categorization.

False Positives, False Negatives, and Human Oversight

LeakScanner has flaws, just like any automated detection method. False negatives, or missed secrets, can give businesses a false sense of security, while false positives, or benign strings marked as secrets, can waste analyst time and cause alert fatigue. These dangers are not eliminated by using an LLM, and if cues or model behavior are misinterpreted, there may be further uncertainty.

The technology is meant to supplement human expertise rather than replace it in order to address these problems. Analysts capable of confirming context, de-duplicating recurring problems, and organizing remediation should prioritize findings. To reflect each tenant's

changing environment, configuration files and regex patterns should be checked and adjusted on a regular basis.

Dual Use and Information Warfare

Tools that can scan for disclosed secrets can be used both offensively and defensively from the standpoint of information warfare. While offensive usage may enable attackers to find critical architecture information or vulnerable credentials at scale, defensive use assists companies in locating and fixing leaks. Although LeakScanner's local-first, configuration-based methodology was created with internal security teams and defensive managed-service providers in mind, it is important to recognize the possibility of dual use.

Deployment that is morally sound entails:

- Restricting Dissemination To Organizations And Users Who Are Trusted.
- Incorporating Explicit Acceptable-Use Clauses Into Documentation And Licensing.
- Putting A Strong Emphasis On Education And Awareness So People Are Aware Of Their Limitations As Well As Their Capabilities.

CONCLUSION AND FUTURE WORK

This work presented **LeakScanner**, a configuration driven, multi tenant leak detection tool that combines GitHub code search, regex-based pattern matching, and optional LLM assisted analysis to identify potential credential and proprietary code leaks. Tenant configuration and the scanning engine are kept apart by the system architecture, allowing many organizations with different brands, domains, and detection criteria to be supported by a single deployment. The prototype includes a test mode that produces realistic mock results for safe demonstrations, secure management of API tokens via environment variables, and per-tenant reporting in Markdown and JSON.

LeakScanner offers a lightweight, local-first design that is suited for managed security providers, SMEs, and educational settings in contrast to other tools and commercial platforms. Onboarding new tenants is made easier by its configuration-centric approach, and the potential inclusion of Gemini models shows how LLMs may supplement conventional pattern-based detection with contextual judgments.

Future research will concentrate on a number of areas. First, adding concurrency (such as asynchronous HTTP requests or thread pools) and more complex rate-limit management for external APIs can enhance the performance of live scans. Second, more source integrations would extend the tool's reach beyond GitHub and bring it into line with actual multi-platform development environments. Examples of these interfaces include GitLab, Bitbucket, and cloud storage providers. Third, a simple web dashboard may be created to show results, monitor the progress of remediation, and grant various stakeholders role-based access. Lastly, user tests involving security analysts and a more thorough assessment of real-world repositories would aid in measuring detection accuracy, false-positive rates, and the usefulness of AI-assisted triage.

In conclusion, **LeakScanner** shows that multi-tenant leak detection may be supported by a rather straightforward but well-designed technology in a way that is comprehensible, expandable, and consistent with information-warfare education and defensive cybersecurity practice.

Bibliography

[1] Z. Y. S. P. Y. Z. Runhan Feng, "Runhan Feng," [Online]. Available: https://aoa0.github.io/pubs/icse22.pdf.

[2] GitGuardian, "GitGuardian(DOCS)," 27 October 2025. [Online]. Available: https://docs.gitguardian.com/secrets-detection/secrets-detection-engine/quick_start.

[3] D. O. Cabasa, A. C. Co, D. J. M. Espinosa, A. C. Malic and J. C. Mag-isa, "Cognizance Journal," October 2025. [Online]. Available: https://cognizancejournal.com/vol5issue10/V5I1029.pdf.

[4] M. R. M. B. R. Michael Meli, "Symposium," [Online]. Available: https://dev.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_04B-3_Meli_paper.pdf.

[5] A. V. K. ,. I. P. M. a. A. A. P. Anton V. Konygin, "scitepress," ENASE, 2023. [Online]. Available: https://www.scitepress.org/Papers/2023/119836/119836.pdf.

[6] S. Anderson, "BlakYaks," 2nd July 2025. [Online]. Available: https://www.blakyaks.com/resources/how-to-detect-and-clean-up-leaked-secrets-in-your-git-repositories.

[7] N. I. M.-A. S. L. W. Md Rayhanur Rahman, "PubMed Central," 17 March 2022. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC8928718/.

[8] S. A. Z. W. S. M. S. R. S. Md Nafiu Rahman, "arxiv," 26 April 2025. [Online]. Available: https://arxiv.org/html/2504.18784v1.

[9] A. a. K. J. H. a. H. N. a. W. D. a. A. Y. a. F. S. Krause, "CISPA," CISPA, 27 July 2023. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/krause.