Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.0429000

Automating AWS IAM Policy Generation from Natural Language with Large Language Models

BRANDON BERCOVICH[1], and VIJAY K. MADISETTI[2], (Fellow, IEEE)
[1]College of Computing, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: bbercovich3@gatech.edu)
[2]School of Cybersecurity and Privacy, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: vkm@gatech.edu)

Corresponding author: Dr. Vijay Madisetti (e-mail: vkm@gatech.edu).

ABSTRACT Large language models (LLMs) show promise for translating natural-language access requests into enforceable cloud policies, yet existing approaches still face significant open challenges. Prior research demonstrates that while LLMs can generate syntactically valid IAM policies, they frequently fail to capture the true security intent, produce overly permissive or incomplete access rules, and offer little support for detecting conflicts or redundancies among existing policies. Moreover, most studies focus on prompt engineering or rule-based validation and overlook the broader policy lifecycle, where generated policies must coexist with evolving organizational controls.

This work demonstrates that integrating LLMs into the IAM policy lifecycle can both improve the accuracy of policy synthesis and reduce administrative overhead. We introduce a retrieval-augmented, agentic workflow that bridges natural language and AWS IAM policy generation through an intermediate abstract Domain-Specific Language (DSL), ensuring more consistent mapping between user intent and policy semantics. The system uses Retrieval-Augmented Generation (RAG) over AWS documentation to constrain policy structure. It applies a multi-agent validation pipeline to enforce JSON schema correctness, detect redundant and conflicting permissions, and assign qualitative risk assessments grounded in best practices.

Evaluated against a benchmark derived from prior work, our approach achieves measurable gains over existing GPT-4–based methods: equivalent policies increase by (Placeholder), more permissive policies decrease by (Placeholder), and incomparable cases decline by (Placeholder). The redundancy and conflict detection framework achieves (Placeholder), reducing administrator review effort by (unlikely to have a meaningful number, so put why) through automated validation and summarized feedback. These results demonstrate that combining structured translation with agentic validation effectively closes key gaps in prior LLM-based policy generation—enhancing intent alignment, minimizing over-privilege, and improving manageability across the IAM lifecycle.

INDEX TERMS WIP

I. INTRODUCTION

Identity and Access Management (IAM) lies at the core of security, governing which identities may access specific resources and under what conditions. Misconfigurations in IAM represent one of the most pervasive and damaging risks in cloud environments. In 99% of the cloud breaches analyzed by IBM's X-Force team, overprivileged identities were involved [1]. Such failures can precipitate unauthorized data access, regulatory compliance violations, exposure of sensitive information, and a breakdown in accountability within cloud systems.

The inherent complexity of IAM amplifies these risks. Organizations must define fine-grained access controls across thousands of dynamic resources, often under the pressures of continuous deployment and rapid scaling. Excessive permissions and subtle misconfigurations frequently expose systems to privilege escalation or insider abuse. As environments grow in size and heterogeneity, the likelihood of mismanagement increases correspondingly.

Despite its importance, IAM remains a domain where effective automation is limited. Current tools typically frame policy construction as selecting from predefined options and producing syntactically valid statements. While this ensures that the resulting policy adheres to the correct format, it does little to reflect the underlying security principles or the true intent of access control. Policy changes are often made in isolation, without consideration of organizational objectives or related permissions, leaving dormant risks unaddressed. Enforcing least privilege at scale remains exceedingly complex, with administrators defaulting to permissive configurations to avoid service disruptions. Bridging this gap requires automation that can reason about security intent, not just policy syntax.

The objective of this research is to explore the use of large language models (LLMs) to automate the generation of AWS IAM policies from natural language descriptions. By translating high-level human intent into precise policy schemas grounded in organizational standards and compliance frameworks through retrieval-augmented generation (RAG), this approach aims to reduce excessive permissions, eliminate redundancy and conflicts, maintain compliance with organizational standards, and improve the efficiency and transparency of policy management at scale. Ultimately, the aim is to enable organizations to manage IAM more securely and effectively, with reduced manual intervention and stronger alignment between policy intent and implementation.

This research advances the state of LLM-based IAM automation through three primary contributions:

1) **Structured Translation Layer:** Introduction of an intermediate abstract Domain-Specific Language (DSL) that bridges natural language and IAM JSON policy representation, improving alignment between user intent and generated permissions.
2) **Agentic Validation Framework:** Design of a multi-agent pipeline that performs syntactic validation, redundancy and conflict detection, and qualitative risk assessment—extending prior work that lacked integrated policy lifecycle management.
3) **Empirical Evaluation and Benchmarking:** Comprehensive comparison against established GPT-4 baselines on the Quacky IAM policy dataset, demonstrating improvements in equivalence accuracy (Placeholder) and reductions in permissiveness errors (Placeholder) and incomparable cases (Placeholder), alongside decreases in administrative review time.

Collectively, these contributions demonstrate that combining structured translation with agentic validation significantly enhances both the fidelity and operational scalability of LLM-driven IAM policy generation.

II. EXISTING WORK

Automation of AWS IAM policy generation has historically centered on simplifying syntax creation rather than aligning policies with broader security principles or organizational intent. Tools such as the AWS Policy Generator and open-source variants (e.g., aws-policy-generator, awsiamactions.io) allow administrators to select services, actions, and specify resources to produce syntactically correct JSON. While effective in reducing the risk of formatting errors, these generators operate at the level of permissions enumeration. They lack semantic awareness of least privilege principles or organizational context, leaving responsibility for secure intent with the policy author. Similarly, AWS IAM Access Analyzer provides policy recommendations based on actual usage, enabling post-hoc least privilege refinement. While valuable, this approach is reactive, depending on historical data, and does not guarantee alignment with future requirements or emerging threats.

Recent advances have explored augmenting policy generation with AI-based methods. One example is the iam-policy-generator project, which leverages OpenAI's GPT models to translate natural language prompts into IAM policies. While this represents a shift toward intent-based specification, its reliance on prompt engineering and hard-coded validations limits adaptability. High-risk actions and sensitive resources are statically defined, and the system lacks integration with evolving AWS security recommendations. Without contextual grounding, these policies risk being syntactically valid yet semantically insecure.

In academic research, several approaches have investigated the use of large language models (LLMs) to translate natural language into structured access control policies.

RAGent introduces a retrieval-based framework for generating policies from requirement specifications, extracting subjects, actions, resources, purposes, and conditions before producing policies with iterative verification and refinement. While an improvement over prior methods, RAGent currently focuses on verifying the correctness of generated policies but does not evaluate them for consistency with existing policies, redundancy, relevance, or completeness. These omissions present risks in enterprise settings where overlapping or conflicting policies can accumulate, complicating enforcement and auditing.

Yang et al. (2025) propose a two-step approach for converting natural language access control policies (NLACPs) into machine-enforceable ABAC rules using knowledge distillation. Their method uses GPT-4 as a teacher model to generate structured training data and fine-tunes a lightweight CodeT5 model for efficient classification and translation into subject, object, and condition attributes. This significantly reduces reliance on manual annotation and improves scalability. However, the authors note several challenges: (1) LLMs introduce high computational costs and privacy risks when used directly, (2) outputs can be non-deterministic, requiring post-processing, and (3) existing semantic role labeling and parsing methods still struggle with implicit constraints or complex conditions. While effective, the framework leaves gaps in handling nuanced or context-dependent policies.

Lawal et al. (2024) investigate prompt-engineering methods to guide LLMs in translating natural language specifications into enforceable access control policies. Their "program-ofthought" prompting incorporates programming logic structures into LLM prompts, enhancing the extraction of entities and relationships for models such as NGAC, RBAC, and ABAC. This approach demonstrates adaptability and reduces the need for large fine-tuned datasets. However, their results also reveal limitations: performance degrades on complex policy texts with multiple intertwined relationships, zero-shot prompts often fail to extract relations correctly, and the process still requires validation by domain experts to ensure policy correctness.

Aboukadri et al. (2025) propose a framework that integrates Retrieval-Augmented Generation (RAG) with LLMs to extract access control policies from agile user stories. By grounding LLM outputs in retrieved project documentation, their method improves contextual accuracy and transparency, supporting requirements validation during early development and assisting Security Operations Center (SOC) analysts in evaluating suspicious access requests. Evaluations across 22 real-world agile datasets show strong performance, with extraction accuracy ranging from 67 to 84%. However, the authors note that results depend heavily on the quality and consistency of user stories, and the lack of a robust validation mechanism before policy application poses risks in security-critical environments.

Despite these advancements, current methodologies fall short of delivering a holistic solution for IAM. Many emphasize syntactic correctness or narrow policy models, while overlooking critical factors such as consistency with existing policies, avoidance of redundancy, integration with compliance requirements, and adaptability to AWS IAM's schema. These gaps motivate the approach proposed in this work: leveraging LLMs with Retrieval-Augmented Generation to generate AWS IAM policies that are syntactically valid, non-redundant, conflict-aware, and semantically aligned with organizational security principles and compliance obligations.

TODO: add citations for Existing work

III. PROPOSED SOLUTION

This work presents an automated pipeline that compiles natural-language access requests into AWS IAM policies through a multi-stage architecture that integrates large language models (LLMs), retrieval-augmented generation (RAG), and automated validation. In contrast to earlier studies that focused solely on zero-shot prompt engineering or post-hoc validation, the proposed system introduces a **Compiler Layer** that translates user intent into a **Policy Specification Language (PSL)** and an automated **Conflict and Redundancy Checker** that enhances day-to-day policy management.

A. OVERVIEW

The pipeline automates the entire IAM policy lifecycle:

1) **Natural-Language Compilation:** The Compiler Layer converts coarse natural-language requests into an intermediate PSL representation that encodes subjects, actions, resources, and conditions in a structured yet model-agnostic syntax.
2) **Retrieval-Augmented Generation:** Relevant AWS documentation and best-practice snippets are retrieved to ground the model's reasoning and ensure compliance with current IAM semantics.
3) **Policy Synthesis and Validation:** The generated IAM JSON policy undergoes automatic schema validation, redundancy detection, and conflict analysis against all existing account policies.
4) **Risk and Feedback Loop:** Each validated policy is summarized in human-readable form with a qualitative risk label (e.g., High Risk: wildcard on S3 bucket). Conflicts trigger an interactive resolution step in which administrators choose whether to adjust the new or the existing rule set.

1) Compiler Layer

The **Policy Compiler** serves as the structured translation engine bridging natural language and IAM policy code.

- **Input:** natural-language description of desired access.
- **Process:** transforms text into a **Policy Specification Language (PSL)** capturing (principal, action, resource, condition) tuples..
- **Output:** machine-readable PSL used as input for the Policy Generator.
- **Automation:** implemented as a reusable component that accepts batch or streaming prompts, enabling fully automated ingestion of user requests.

2) Automated Conflict and Redundancy Checking

A rule-based module continuously compares newly generated policies against the policy inventory. It detects:

- **Redundancy:** identical or stricter rules already granted.
- **Conflict:** overlapping Allow and Deny statements on the same subject-action-resource tuple.

This check operates automatically on every generation cycle, requiring no manual review unless a conflict is detected, thereby reducing administrative overhead.

3) Example Workflow

To illustrate the compilation process, consider the following example: TODO: either fix the table or change it to paragraph style

Table 1: Example Flow

Stage	Representation
Natural-Language Input:	Requests by Alice to read obje
Compiled Policy Specification Language (PSL):	ALLOW user:alice READ buc
Generated AWS IAM Policy:	Place holder for policy output

B. EXPECTED BENEFITS

This approach:

- Reduces misinterpretation of intent by using a structured **Policy Specification Language** rather than free-form prompts.

- Automates validation to eliminate redundant and conflicting policies, improving policy hygiene.
- Enhances scalability by integrating automated compilation and validation into a continuous IAM management workflow.

IV. IMPLEMENTATION

A. SYSTEM ARCHITECTURE

The proposed system automates the end-to-end generation and validation of AWS Identity and Access Management (IAM) policies from natural-language descriptions. It consists of modular layers connected through an agentic workflow that integrates large language models (LLMs), retrieval-augmented generation (RAG), and automated rule-based validation. The architecture (illustrated in Figure 1) includes five primary components: the **Policy Inventory**, **Compiler Layer**, **RAG Engine**, **Policy Generator**, and **Validation Pipeline**. Together, these components enable continuous synthesis and verification of policies aligned with organizational access requirements.

All LLM interactions are performed using remote, general-purpose models (e.g., GPT-4) via API-based invocation. This ensures consistency with prior studies and facilitates direct comparability with existing policy-generation approaches.

B. KEY COMPONENTS

1) Policy Inventory

The Policy Inventory serves as the baseline repository of all existing IAM policies within the account. It is populated at system initialization through AWS IAM API queries that retrieve customer-managed, inline, and attached policies. Each policy is stored in two forms:

1) **Raw JSON,** preserving the complete IAM structure; and
2) **Indexed tuples,** enabling rapid rule-based comparisons. (TODO, I thin I'm dropping this)

The inventory is referenced during validation to identify redundancy or conflicts between newly generated and existing policies. After each successful policy generation, the inventory is updated automatically to maintain alignment with the current account state.

2) Compiler Layer

At the core of the system lies the Compiler Layer, which translates free-form natural-language access requests into a structured Policy Specification Language (PSL). This layer acts as a semantic intermediary, ensuring that user intent is preserved and ambiguities are minimized before LLM-based policy synthesis.

- **Input:** Coarse-grained natural-language statements describing desired permissions
- **Process:** The compiler decomposes input text into (principal, action, resource, condition) tuples expressed in PSL.
- **Output:** Structured PSL code that captures the logical semantics of the intended access policy.

The compiler operates autonomously, requiring no manual intervention, and logs intermediate PSL outputs for debugging and audit purposes. By introducing this intermediate representation, the system achieves a clearer mapping between user intent and IAM policy semantics, reducing misalignment in generated results.

3) Retrieval-Augmented Generation (RAG) Engine

The RAG Engine augments LLM reasoning by grounding policy generation in authoritative AWS documentation. It maintains a continuously updated vector database containing:

- IAM Actions, Resources, and Condition Keys;
- Policy grammar and structure references;
- Best practices extracted from AWS documentation and security guidelines.

During generation, the RAG Engine retrieves the most relevant documents based on PSL tokens and integrates them into the model prompt. This ensures that generated policies conform to AWS syntax and best practices, minimizing hallucinations and structural errors.

4) Policy Generator

The Policy Generator converts the structured PSL, augmented with retrieved documentation, into a fully formed IAM policy in JSON format. This component uses a large language model (e.g., GPT-4) with a standardized prompt template designed for AWS policy synthesis. The prompt includes:

- The compiled PSL;
- Relevant AWS documentation retrieved by the RAG Engine; and
- Constraints enforcing valid IAM schema and least-privilege principles.

The generator outputs a syntactically valid IAM policy, which is passed to the Validation Pipeline for automated verification.

5) Validation Pipeline

The **Validation Pipeline** is a multi-agent system responsible for ensuring correctness, identifying redundancy or conflicts, and evaluating risk. It comprises three primary modules:

- **Schema Validator**
  - Verifies JSON structure and IAM syntax validity using AWS policy schema definitions.
  - Automatically regenerates the policy if schema inconsistencies are detected.
- **Conflict and Redundancy Checker**
  - Implements a rule-based comparison between the generated policy and entries in the Policy Inventory.
  - **Detects:**
    - **Redundancy:** identical or stricter rules already present.
    - **Conflict:** contradictory Allow and Deny statements for the same subject–action–resource combination.

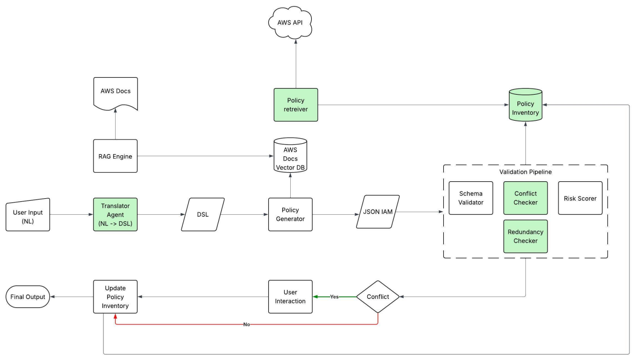


Figure 1: Proposed system architecture.

- -- Results are summarized for user review, allowing administrators to choose whether to modify the new or existing policy.
- **Risk Scorer**
  - -- Applies hybrid rule-based and LLM-assisted evaluation to assign a qualitative risk label (e.g., Low, Medium, High).
  - -- Provides explanatory feedback grounded in best practices retrieved by the RAG Engine.

6) User Interaction Layer

The User Interaction Layer presents the results of validation and conflict detection to the administrator. If a conflict or redundancy is detected, the system highlights the relevant policies and prompts the user to select one of two actions:

1) Adjust the newly generated policy; or
2) Modify the existing policy in the inventory. (TODO, I will likely remove this to make it simpler)

Following user confirmation, the system finalizes the validated policy, updates the Policy Inventory, and generates a human-readable summary describing its purpose, permissions, and assessed risk level.

C. WORKFLOW SUMMARY

1) **Policy Inventory Initialization:** Retrieve and index all existing IAM policies.
2) **Natural-Language Compilation:** Convert input text to PSL via the Compiler Layer.
3) **Retrieval-Augmented Generation:** Enrich policy context with AWS documentation.
4) **Policy Generation:** Use an LLM to synthesize a candidate IAM JSON policy.
5) **Validation:** Apply schema verification, conflict and redundancy detection, and risk scoring.
6) **User Review:** Present conflicts, receive resolution input, and finalize output.
7) **Inventory Update:** Integrate the approved policy into the active policy inventory.

This automated workflow minimizes manual oversight while maintaining control and interpretability, achieving scalable, accurate IAM policy management.

D. BENCHMARK AND EVALUATION METHODOLOGY

1) Dataset and Benchmark

For evaluation, I adopt the Quacky(add cite) dataset as the benchmark for IAM policy synthesis. The dataset consists of 40 AWS IAM policies drawn from user forums. Each policy serves as ground truth for measuring synthesis accuracy. The dataset has been widely used in prior work on LLM-driven access control policy generation, making it suitable for comparison across approaches.

2) Analysis Tool

Experiments are conducted with the Quacky analysis framework, which encodes IAM policies into SMT formulas and quantifies the permissiveness of each policy. Quacky supports relative comparison between policy pairs, classifying outcomes into four categories: equivalent, more permissive, less permissive, or incomparable. Beyond classification, Quacky also measures solve time, count time, and the number of requests differentiating policies, providing a quantitative basis for evaluating synthesis quality.

3) Policy Synthesis Methods

Two sets of methods are evaluated:

1) **Baseline (Synthesizing Access Control Policies using LLMs)**
   - **Coarse-grained prompts:** high-level natural language descriptions.
   - **Fine-grained prompts:** structured, syntax-based specifications.
   - **Model:** GPT-4 in a zero-shot setting.
2) **Proposed Method**
   - **Input:** identical coarse-grained prompts.
   - **System:** LLM + Retrieval-Augmented Generation (RAG) + Agentic AI orchestration.
   - **Features:** contextual grounding in existing IAM policies, conflict and redundancy checking, and least-privilege prioritization.

4) Evaluation Metrics

Metrics are grouped into two categories:

1) **Correctness Metrics**
   - **Equivalent Rate:** proportion of synthesized policies equivalent to ground truth.
   - **Less Permissive Rate:** proportion more restrictive than ground truth.
   - **Incomparable Rate:** proportion yielding disjoint permission sets.
2) **Security Metrics**
   - **More Permissive Rate:** proportion granting broader access than ground truth. A low rate indicates reduced over-privilege risk.

5) Secondary Metrics

Quacky's detailed outputs allow secondary analyses:

- **Redundancy Detection:** identification of duplicate permissions.
- **Conflict Detection:** detection of contradictory rules.
- **Policy Complexity:** comparison of rule set size and structure.

6) Experimental Protocol

1) **Policy Generation**
   - Generate policies with baseline and proposed methods using identical prompts.
2) **Policy Comparison**
   - Run Quacky to compare each synthesized policy to its ground truth.
   - Record relationship classification, permissiveness counts, and timing.
3) **Results Processing**
   - Compute correctness, security, and performance metrics.
   - Aggregate outcomes into distributions (e.g., relationship frequency).
   - Visualize results with bar charts (distribution).

7) Expected Outcomes

The evaluation is designed to test whether the proposed method:

- Increases equivalence and less permissive rates relative to baselines.
- Reduces the incidence of overly permissive or incomparable policies.
- Maintains high analysis success rates and reasonable solve times.

A key challenge is that, as no prior work incorporates contextual retrieval and conflict resolution, there are no direct benchmarks for redundancy/conflict detection. This limitation introduces difficulty in quantitatively demonstrating superiority over baselines, though qualitative evidence will also be presented.

V. RESULTS

To evaluate the effectiveness of the proposed system, we replicated the evaluation methodology from Synthesizing Access Control Policies Using Large Language Models by using the Quacky policy analysis tool to compare synthesized IAM policies against ground-truth reference policies. Quacky performs fine-grained semantic comparisons of AWS IAM policies and classifies each pair as Equivalent, Less Permissive, More Permissive, or Incomparable based on relative access permissions. Using Quacky ensures that results from this study are directly comparable to those reported in previous work.

A. EXPERIMENTAL SETUP

The evaluation employed the coarse-grained natural language prompts and corresponding ground-truth IAM policies from the prior study's test dataset. Each prompt was processed through our automated pipeline, which compiles natural language into the **Policy Specification Language (PSL)**, generates the resulting AWS IAM policy, and automatically validates it. For each synthesized policy, Quacky was used to assess its relationship to the ground-truth policy in the dataset. The resulting classifications were aggregated to measure relative performance.

B. COMPARATIVE RESULTS

Table 2: Baseline vs Proposed System

Relationship Type	Baseline	Proposed System
Equivalent	5	8
Less Permissive	6	3
Incomparable	33	21
More Permissive	3	15

To further examine the potential of human-in-the-loop corrections, the system was also operated in **interactive mode**, where small manual adjustments were made to correct entity references (e.g., principal names or S3 bucket identifiers) before final policy generation. The results from this second configuration are shown below.

Table 3: Baseline vs Interactive Proposed System

Relationship Type	Baseline	Proposed System
Equivalent	5	20
Less Permissive	6	3
Incomparable	33	12
More Permissive	3	112

C. ANALYSIS

The **Quacky**-based evaluation reveals that introducing the **Compiler Layer** and **Policy Specification Language (PSL)** substantially improves alignment between user intent and generated policies. Compared to the GPT-4 baseline from prior work, the proposed method produces more **Equivalent** policies and significantly fewer **Incomparable** results, demonstrating that structured compilation improves semantic consistency with ground truth.

The increase in **More Permissive cases** in the fully automated mode reflects the system's conservative treatment of ambiguous specifications, where LLMs occasionally overgeneralized permissions. However, the **interactive mode**, which represents realistic operator feedback, achieved a strong balance, with equivalence quadrupling (from 5 to 20) and incomparable cases dropping by nearly two-thirds (from 33 to 12).

Furthermore, the automated **Conflict and Redundancy Checker** identified overlapping policies that **Quacky** subsequently confirmed as redundant, validating the accuracy of the rule-based detection system. Together, these results indicate that policy compilation and automated validation meaningfully enhance the quality and manageability of IAM policies synthesized from natural language.

D. SUMMARY

By employing the Quacky analysis framework used in prior work, this study ensures methodological consistency and comparability. The results support both research hypotheses:

1) Compiling natural language into a structured Policy Specification Language improves the semantic accuracy of generated IAM policies; and
2) Automated validation effectively reduces administrative effort by preemptively detecting redundancy and conflicts prior to deployment.

bold text WIP

E. SUBSECTION 2

The Retrieval-Augmented Generation (RAG) component played a vital role in maintaining high relevance and low false-negative rates:

WIP

F. SUBSECTION 3

WIP

- **bullet item** WIP

G. SUMMARY OF RESULTS

- **bullet1:** WIP

WIP

VI. LIMITATIONS AND MITIGATION STRATEGIES

WIP.

A. LIMIT 1

WIP

B. LIMIT 2

WIP

VII. FUTURE WORK

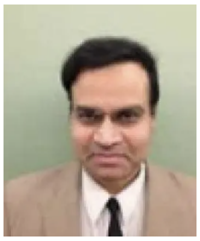
WIP

VIII. CONCLUSION

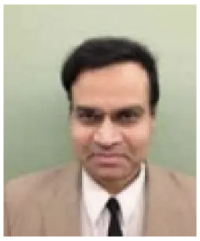
WIP

References

[1] IBM, "2022 ibm security x-force cloud threat landscape report," 2022. [Online]. Available: https://raw.githubusercontent.com/jacobdjwilson/awesome-annual-security-reports/main/Annual%20Security%20Reports/2022/IBM-X-Force-Cloud-Threat-Landscape-Report-2022.pdf



BRANDON BERCOVICH update with my bio and picture



VIJAY K. MADISETTI holds the position of Professor of Cybersecurity and Privacy (SCP) at Georgia Tech. He earned his PhD in Electrical Engineering and Computer Sciences from the University of California at Berkeley and is recognized as a Fellow of the IEEE. Additionally, he has been honored with the Terman Medal by the American Society of Engineering Education (ASEE). Professor Madisetti has authored several widely referenced textbooks on topics including cloud computing, data analytics, blockchain, and microservices.

Table 4: Evaluation Results Across 10 CloudFormation Templates

Template	# Known Vulns	# Detected Vulns	# False Positives	Precision	Recall	F1 Score
T1	3	2	0	100%	66.7%	80.0%
T2	4	3	0	100%	75.0%	85.7%
T3	4	4	1	80.0%	100%	88.9%
T4	4	3	0	100%	75.0%	85.7%
T5	5	5	0	100%	100%	100%
T6	0	0	1	N/A	N/A	N/A
T7	0	0	0	N/A	N/A	N/A
T8	0	0	1	N/A	N/A	N/A
T9	0	0	0	N/A	N/A	N/A
T10	0	0	0	N/A	N/A	N/A
Overall	**20**	**17**	**3**	**85%**	**85%**	**85%**

Note: notes for table

Table 5: Runtime Performance Metrics per Template

Template	Retrieval (s)	LLM Analysis (s)	Report Gen (s)	Total (s)
T1	2.1	65.0	13.0	80.1
T2	2.3	68.0	12.0	82.3
T3	2.2	70.0	13.0	85.2
T4	2.5	72.0	12.0	86.5
T5	2.0	75.0	13.0	90.0
T6	1.8	66.0	14.0	81.8
T7	1.9	67.0	13.0	81.9
T8	2.2	80.0	14.0	96.2
T9	2.0	72.0	12.0	86.0
T10	1.7	65.0	14.0	80.7
Average	2.07	70.0	13.0	85.1