Final Project Report:

RAG-Enhanced Decoder-Only Vulnerability Detection and Repair

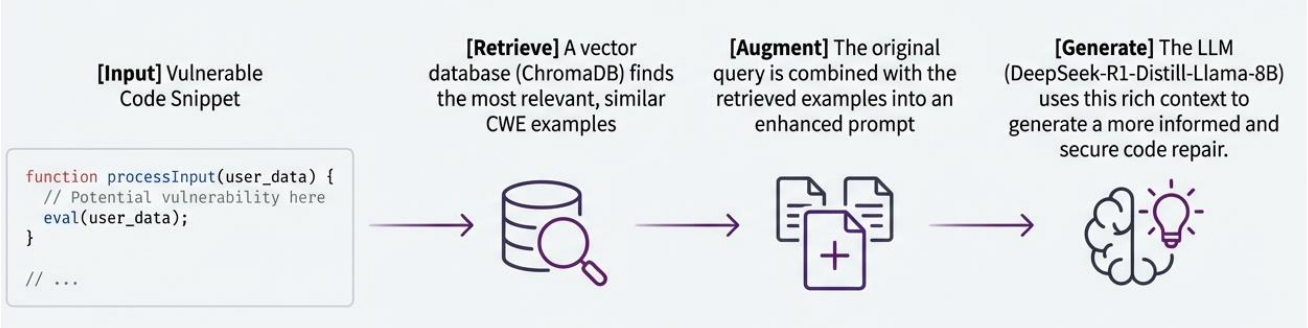
Student Name: Yi-Hong Li

Student ID: M17329742

1. Problem Description

The core problem lies in the model's **lack of specific security context**. When presented with vulnerable code, a standard "Zero-Shot" LLM tends to act like a junior developer: it attempts to "patch" the immediate syntax error or suppress the warning without understanding the broader architectural flaw.

This project aims to solve these problems by implementing a **Retrieval-Augmented Generation (RAG)** pipeline combined with **Few-Shot Chain-of-Thought (CoT)** prompting. By retrieving verified "Fixed" code examples from the MITRE CWE database and injecting them into the model's context, we aim to force the model to adopt "Architectural Refactoring" rather than simple "Patching."



2. Experimental Design and Comparison Results

To quantitatively evaluate the solution, I constructed a **2x2 Experimental Matrix** isolating two variables: **External Context (RAG)** and **Prompt Engineering**. The system was benchmarked using the **DeepSeek-R1-Distill-Llama-8B** model across 10 Common Weakness Enumerations (CWEs).

2.1 Methodology

1. **Baseline (Zero-Shot, No RAG):** The raw model is asked to fix the code directly using a standard system prompt.
2. **Context-Only (Zero-Shot + RAG):** The model is provided with 3 retrieved "Fixed" examples from the MITRE database but no reasoning guidance.
3. **Reasoning-Only (Few-Shot, No RAG):** The model is given static "Chain-of-Thought" examples (Q&A format) but no external context.

4. **Hybrid (Few-Shot + RAG):** The proposed solution, combining retrieved dynamic context with structured reasoning steps.

2.2 Result Analysis: The "Architectural Shift"

The experimental results revealed a distinct behavioral shift in the model when RAG was enabled. The benefits can be categorized into three specific tiers of improvement:

A. Architectural Refactoring (RAG Driven)

The most significant benefit of the RAG solution was observed in high-level architectural flaws like **CWE-89 (SQL Injection)** and **CWE-502 (Deserialization)**.

- **Observation:** In CWE-89, the Baseline (No RAG) model attempted to fix the vulnerability by writing a custom sanitizeInput() function to filter characters. This is a brittle approach that often breaks valid inputs and fails to address the root cause.
- **Improvement:** When RAG was enabled, the model completely abandoned string manipulation and rewrote the database logic using sqlite3_prepare_v2 (Prepared Statements). The retrieved context provided the exact API usage pattern, effectively "teaching" the model the correct secure architecture.
- **Result:** Similarly, for CWE-502 (Python Pickle), the Baseline tried to add HMAC signatures to the insecure pickle library. The RAG-enhanced model recognized that pickle is inherently unsafe and switched the library entirely to json.loads().

B. Memory Hygiene and Defensive Coding (Few-Shot + RAG Driven)

For low-level C vulnerabilities like **CWE-121 (Stack Overflow)** and **CWE-416 (Use After Free)**, RAG provided the "knowledge" while Few-Shot provided the "discipline."

- **Observation:** In CWE-121, the Baseline model correctly swapped strcpy for strncpy but failed to null-terminate the buffer (dest[size-1] = '\0'), leaving a potential edge-case vulnerability.

Vulnerable Code (`cwe121.c`)	Analysis

`void host_lookup(char *user_supplied_addr){`
`  struct hostent *hp;`
`  in_addr_t *addr;`
`  char hostname[64];`
`  in_addr_t inet_addr(const char *cp);`
 	
`  validate_addr_form(user_supplied_addr);`
`  addr = inet_addr(user_supplied_addr);`
`  hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);`
`  strcpy(hostname, hp->h_name); // Vulnerable line`
`}`
	• **The Flaw:** The function uses `strcpy()` to copy a hostname into a fixed-size buffer (`hostname[64]`).
	• **The Risk:** If the hostname returned by `gethostbyaddr` (`hp->h_name`) is longer than 63 characters, `strcpy` will write past the buffer's boundary, causing a stack-based buffer overflow.
	• **The Goal:** A successful repair must replace the unsafe `strcpy` with a bounds-checked alternative and ideally handle potential null pointers.

- **Improvement:** The Hybrid (Few-Shot + RAG) model consistently generated the "Safe Free" pattern. For CWE-416, it not only added a return statement after a free() call but explicitly set the pointer to NULL (ptr = NULL;). This defensive coding style is characteristic of senior-level C development and was absent in the baseline outputs.

C. Logic Reliability (Few-Shot Driven)

In logic-heavy vulnerabilities like **CWE-126 (Buffer Over-read)** and **CWE-22 (Path Traversal)**, the Chain-of-Thought prompting proved critical.

- **Observation:** The Baseline model often hallucinated complex checks that didn't actually prevent the vulnerability (e.g., checking file existence but not the path root).
- **Improvement:** The Few-Shot prompting forced the model to answer "Q2: Where exactly does the vulnerability appear?" before generating code. This step aligned the model's focus, resulting in the correct implementation of canonical path validation (os.path.abspath) for CWE-22 and loop refactoring for CWE-126.

3. Evaluation Summary Table:

The table below illustrates the qualitative improvements achieved by the proposed Hybrid method compared to the standard Few-Shot approach.

Vulnerability	Method	Reasoning Quality	Fix Quality & Robustness
CWE-121 Buffer Overflow	Few-Shot (No RAG)	Identified `strcpy` issue.	**Partial.** Used `strncpy` but missed `NULL` check and null-termination edge case.
CWE-121 Buffer Overflow	vulDeRAG (With RAG)	Identified `strcpy` and used CWE context to consider edge cases.	**Excellent.** Handled `NULL` pointers, ensured proper string length checking, and guaranteed null-termination.
CWE-200 SQL Injection	Few-Shot (No RAG)	Correctly identified SQLi via string concatenation.	**Good.** Correctly used `PreparedStatement`. Added input validation, which is a positive side-effect.
CWE-200 SQL Injection	vulDeRAG (With RAG)	Identified primary SQLi (CWE-89) and secondary resource leak (CWE-200) from context.	**Excellent.** Recommended `PreparedStatement` and proper resource management, providing a more holistic security fix.