CyberSecure-Learn - A Local RAG-LLM Tutor for Secure Coding and Cybersecurity Education

Abstract

This report documents the "CyberSecure-Learn" project, an MVP (Minimum Viable Product) for a local-first cybersecurity education system. The system leverages Retrieval-Augmented Generation (RAG) capabilities combined with a locally-hosted Large Language Model (LLM), specifically Qwen-3 14B, to provide an interactive, chat-based tutoring experience. Its core functionality involves answering user queries on secure coding and general cybersecurity topics by retrieving relevant information from a comprehensive PDF corpus of over 100 cybersecurity books. The project emphasizes offline operation, ethical AI practices, robust source attribution, and mechanisms for long-context learning and secure code generation, making it a valuable tool for self-paced cybersecurity education.

1. Project Objective and Architecture

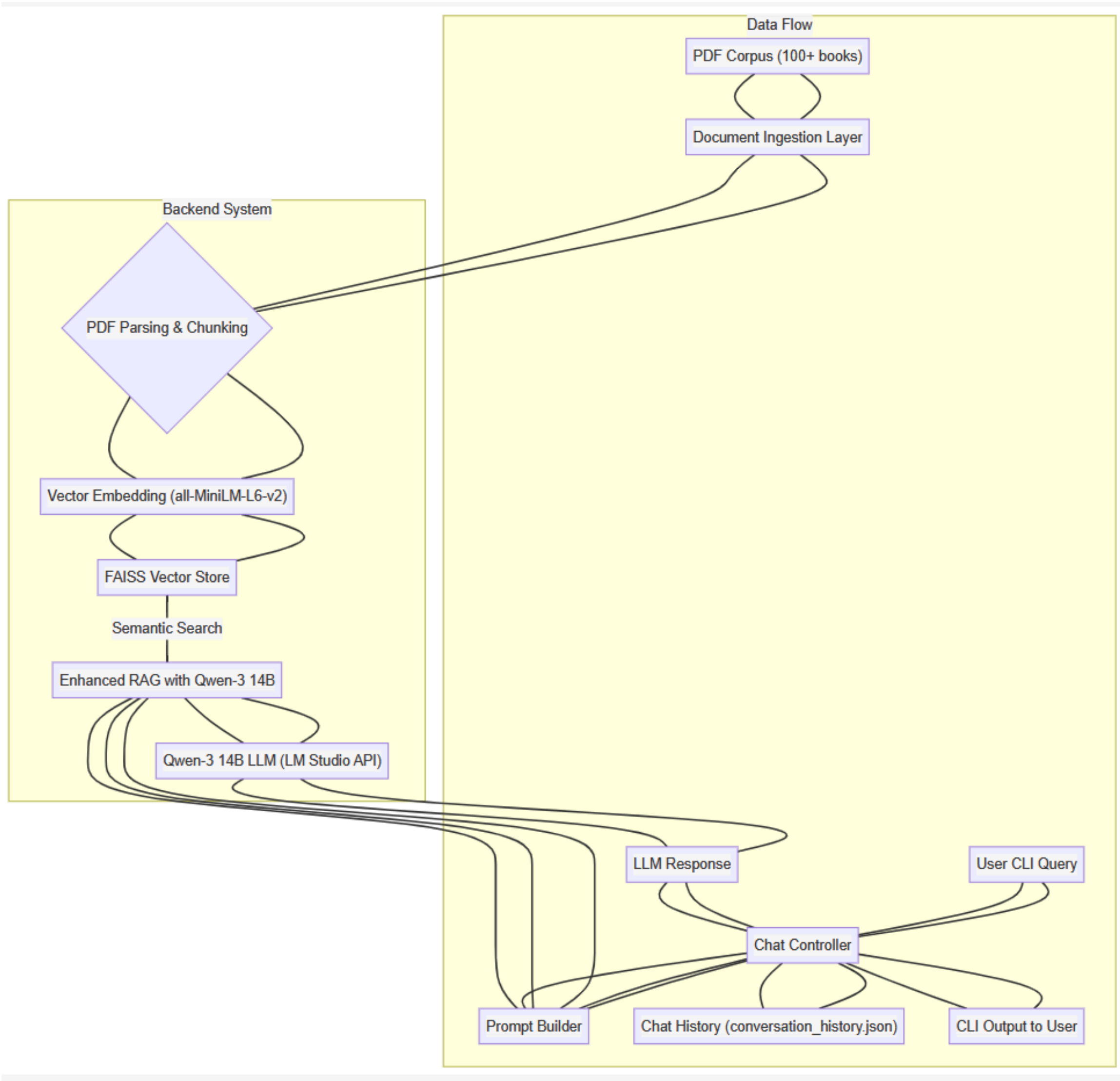
1.1 Project Objective

The primary objective of CyberSecure-Learn is to create an accessible, effective, and privacy-preserving educational tool for cybersecurity. By combining a local LLM with a RAG pipeline, the project aims to:

- Provide accurate and contextually relevant answers to cybersecurity questions.
- Enable learning from a vast corpus of specialized PDF documents.
- Support the generation of secure code examples and explanations.
- Ensure ethical AI responses with explicit source attribution and confidence metrics.
- Operate entirely offline, ensuring data privacy and accessibility without internet dependency.

1.2 System Architecture Overview

CyberSecure-Learn is designed as a local-first system, meaning all components and data processing occur on the user's local machine, ensuring privacy and offline functionality. The architecture is modular, comprising distinct layers that handle document ingestion, data indexing, information retrieval, LLM interaction, and user interface management.



Architectural Layers:

- **Document Ingestion Layer:** Responsible for processing the raw PDF corpus.
- **Vector Embedding + FAISS-based Semantic Search:** Transforms textual data into numerical vectors for efficient similarity search.
- **RAG-based Prompt Builder and LLM Wrapper (Qwen-3 14B):** Orchestrates the retrieval process and constructs the final prompt for the LLM.
- **Chat Controller with Memory and Source Tracking:** Manages the conversational flow, maintains chat history, and handles output formatting.
- **CLI Chatbot Interface:** Provides the user-facing command-line interaction.

2. Prerequisites & Setup

Follow the steps below to replicate or run CyberSecure-Learn from scratch:

2.1 Python Environment & Dependencies

Set up a Python virtual environment, activate it, and install required dependencies from the provided `requirements.txt` file.

2.2 Install LM Studio (LLM Runner)

Install LM Studio from https://lmstudio.ai. After installation:

- Launch the application.
- Download the "Qwen-3 14B Chat" model.
- Enable the OpenAI-compatible API server on port 1234 (default).

  The system is configured to connect to this server at http://127.0.0.1:1234/v1.

2.3 Install Ollama (Supporting Layer)

Install Ollama from https://ollama.com. It acts as a required backend layer to support execution of large local models like Qwen-3 14B. In some configurations, it may automatically start in the background and facilitate compatibility for LM Studio to run the model smoothly.

Do not skip installing Ollama, as it is necessary in many local LLM setups even if you're using LM Studio as your main interface.

2.4 Organize Your PDFs

Place all cybersecurity books, references, or training material PDFs in a folder named `pdfs`. This folder must be in the same directory as the chatbot script (`rag_llm_1.py`), as it is used for document retrieval.

2.5 Run the Chatbot

To start the system:

- Ensure LM Studio is running and the Qwen model is loaded.
- Activate your virtual environment (if applicable).
- Run the chatbot in interactive mode using the following command:

```
python rag_llm_1.py --mode interactive
```

3. Backend System: LLM + RAG Implementation

The core of CyberSecure-Learn's intelligence lies in its sophisticated RAG + LLM fusion, enabling it to retrieve precise information and generate coherent, contextually relevant responses.

3.1 Tech Stack

The project is built entirely on Python, leveraging robust libraries for each critical component:

- **Programming Language:** Python
- **LLM:** Qwen-3 14B, hosted locally via the LM Studio API
- **Vector Search:** FAISS (Facebook AI Similarity Search)
- **Embeddings:** SentenceTransformer (`all-MiniLM-L6-v2`)

- **PDF Parsing:** PyMuPDF (`fitz`)
- **Chunking:** NLTK-based adaptive chunker
- **History Management:** JSON for multi-session chat logging
- **Key Libraries:** `requests` , `nltk` , `torch` , `faiss-cpu` , `sentence-transformers` , `openai` (used for LM Studio API compatibility)

3.2 PDF Processing and Chunking

The document ingestion layer is crucial for preparing the extensive PDF corpus for effective retrieval.

- **PDF Parsing:** PyMuPDF (imported as `fitz`) is used to robustly extract text from PDF documents. This library handles various PDF structures and layouts, ensuring maximum text extraction fidelity. Error handling is implemented to gracefully manage corrupted or unreadable PDF files during this process.
- **Text Cleaning:** Extracted text undergoes cleaning to remove extraneous characters, headers, footers, and other noise that might interfere with embedding quality.
- **Adaptive Chunking:** An NLTK-based adaptive chunker processes the cleaned text. Instead of fixed-size chunks, this chunker dynamically splits content into variable-length segments. This approach considers sentence complexity and semantic boundaries, ensuring that each chunk maintains a coherent context. An overlap logic is applied between consecutive chunks to preserve continuity and reduce the risk of losing context at chunk boundaries.

3.3 Vector Embedding and FAISS Indexing

- **Embedding Generation:** Each processed text chunk is transformed into a high-dimensional numerical vector using the `SentenceTransformer` model (`all-MiniLM-L6-v2`). This model is chosen for its efficiency and effectiveness in generating semantically rich embeddings suitable for similarity search.
- **FAISS Indexing:** The generated embeddings are stored in a FAISS (Facebook AI Similarity Search) index. FAISS is optimized for similarity search on large datasets, allowing for extremely fast retrieval of the most relevant chunks based on a query's embedding. The index is persistent, meaning it can be loaded and saved, avoiding re-embedding the entire corpus on every run.

3.4 RAG-based Prompt Builder and LLM Wrapper

The Retrieval-Augmented Generation (RAG) mechanism is central to the system's ability to provide accurate and cited answers.

- **Query Embedding:** Upon receiving a user query, it is first embedded using the same `SentenceTransformer` model used for the document chunks.
- **Relevant Chunk Retrieval:** The embedded query is then used to perform a semantic search against the FAISS index. This retrieves the `top-k` most semantically similar chunks from the PDF corpus.
- **Re-ranking:** To enhance relevance, retrieved chunks are re-ranked. This process combines semantic similarity scores from FAISS with a keyword overlap bonus, ensuring that chunks containing direct keyword matches from the user's query are prioritized.

- **Contextual Prompt Construction:** The `top-k` re-ranked chunks form the core context for the LLM. These chunks are meticulously formatted and appended to the LLM's input prompt. Crucially, inline citations are added to each piece of retrieved context, indicating the source document and chunk ID (e.g., `[Source 2: bookname.pdf - Chunk ID: 123]`).
- **LLM Integration (Qwen-3 14B):** The Qwen-3 14B model, hosted locally via LM Studio API, serves as the primary generative component. The system leverages the `openai` library for API compatibility with LM Studio, facilitating seamless interaction. Failover logic is implemented to handle potential connection issues or errors with the LM Studio endpoint, ensuring system robustness.
- **System Prompt:** A carefully crafted system prompt is prepended to the LLM's input. This prompt instructs Qwen-3 14B to:
    - Adopt an educational and helpful tone.
    - Prioritize ethical guidance and avoid harmful or inappropriate content.
    - Explicitly use and cite the provided context.
    - Generate secure code examples when relevant.
    - Refrain from hallucinating or providing information not supported by the retrieved context.

4. Chat Memory and Continuity

Maintaining conversational flow and context is vital for an effective tutoring experience.

4.1 JSON-based Multi-session Chat Logging

- **`conversation_history.json`:** All conversational data, including user questions, system answers, retrieved sources, timestamps, and confidence scores, is persistently stored in a `conversation_history.json` file. This allows for multi-session continuity.
- **Memory Capping and Summarization:** To manage the LLM's context window efficiently and prevent it from becoming excessively long, the chat memory is capped. Typically, the last 5 turns of the conversation are retained. For longer conversations, previous turns are intelligently re-summarized into a compact context, preserving key information without consuming excessive token limits.
- **Session Switching:** The CLI interface provides functionality for users to switch between different conversation sessions, enabling them to resume previous discussions or start new ones.

5. How Questions are Answered

The process of answering a user's question involves a sophisticated interplay between retrieval and generation:

1. **User Query:** The user enters a question via the CLI.
2. **Context Retrieval:** The `ImprovedRAGWithQwen14B` module takes the query, embeds it, and searches the FAISS index for the most relevant document chunks.
3. **Prompt Construction:** The `Prompt Builder` combines the retrieved chunks (with inline citations), the current chat history (capped and summarized), and the system prompt to create a

comprehensive input for Qwen-3 14B.

4. **LLM Generation:** Qwen-3 14B processes this rich prompt and generates an answer, drawing heavily from the provided context.
5. **Secure Code Generation:** If the query pertains to secure coding practices, the LLM is guided by the system prompt to generate relevant code snippets, highlighting secure patterns and potential vulnerabilities to avoid.
6. **Source Attribution and Confidence:** Before presenting the answer to the user, the system extracts the sources used by the LLM and calculates a confidence score.
7. **Output to User:** The final answer, along with its confidence score and a list of attributed sources, is displayed in the CLI.

5.1 Long-Context Learning and Secure Code Generation

The system is designed to support long-context learning through:

- **Adaptive Chunking:** Ensuring that individual chunks are semantically rich but manageable.
- **Re-ranking:** Prioritizing the most relevant information to be included in the LLM's context window.
- **Memory Summarization:** Efficiently compressing past conversation history.
- **Qwen-3 14B's Context Window:** Leveraging the large context window of Qwen-3 14B to accommodate extensive retrieved content and conversation history.

For secure code generation, the system prompt explicitly guides the LLM to generate code examples that demonstrate secure coding principles, highlight common vulnerabilities (e.g., SQL injection, XSS), and suggest best practices for mitigation.

6. Ethical Filtering, Scoring, Source Attribution, and Confidence Metrics

6.1 Ethical Filtering and Guidance

The system prompt is the primary mechanism for ethical filtering. It provides explicit instructions to Qwen-3 14B to:

- Maintain an educational, non-biased, and objective tone.
- Refuse to generate harmful, illegal, or unethical content.
- Prioritize user safety and responsible AI principles.
- Focus strictly on cybersecurity education and secure coding.

6.2 Source Attribution

Every answer generated by CyberSecure-Learn is accompanied by precise source attribution. This is achieved by:

- **Inline Citations:** During prompt construction, each piece of retrieved text context is augmented with an inline citation format (e.g., `[Source 2: bookname.pdf]`). The LLM is instructed to use these inline citations in its responses.

- **Source Listing:** A comprehensive list of all source documents (book names, PDF filenames) and their specific chunk IDs that contributed to the answer is presented below the generated response.

6.3 Confidence Scoring

Each answer includes a calculated confidence score, providing the user with an indication of the system's certainty in its response. The score is a composite metric based on:

- **FAISS Similarity:** The average similarity score of the top-k retrieved chunks to the user's query. Higher similarity indicates stronger relevance.
- **Response Length:** Longer, more comprehensive responses that are well-supported by retrieved context tend to have higher confidence.
- **Source Diversity:** Answers drawing from multiple distinct sources (different books or different parts of the same book) are considered more robust and thus assigned higher confidence.

6.4 Response Persistence

Responses exceeding 1000 characters in length are automatically saved to disk as `response_{timestamp}.txt` files. This ensures that extensive explanations or code examples are not truncated and are easily accessible for later review.

7. Module Descriptions

This section details the key Python classes and their functionalities within the CyberSecure-Learn project.

7.1 `ConfigManager`

- **Purpose:** Manages application configuration, including LLM endpoint URLs, API keys (if applicable), embedding model paths, FAISS index paths, and chat history file locations. Provides a centralized way to access and modify system settings.
- **Key Functionality:** Loads configurations from a designated file, validates necessary parameters, and exposes settings as attributes.

7.2 `OptimizedPDFProcessor`

- **Purpose:** Handles the robust parsing and initial cleaning of PDF documents.
- **Key Functionality:**
    - Takes a PDF file path as input.
    - Uses PyMuPDF (`fitz`) to extract text page by page.
    - Implements text cleaning routines (e.g., removing multiple spaces, hyphens, non-printable characters).
    - Manages error handling for corrupted or password-protected PDFs.
    - Returns cleaned text content for further processing.

7.3 `AdaptiveChunker`

- **Purpose:** Splits raw text content into semantically coherent, variable-length chunks suitable for embedding and retrieval.
- **Key Functionality:**
  - Utilizes NLTK for sentence tokenization.
  - Implements logic to create chunks based on sentence complexity, aiming for complete ideas within each chunk.
  - Applies an overlap strategy (e.g., a few sentences or words) between consecutive chunks to ensure context is not lost at chunk boundaries.
  - Returns a list of chunked text segments.

7.4 `EnhancedVectorStore`

- **Purpose:** Manages the creation, persistence, and querying of the FAISS vector index.
- **Key Functionality:**
  - Initializes the `SentenceTransformer` embedding model.
  - Takes processed text chunks and generates their embeddings.
  - Builds or loads a FAISS index from disk.
  - Provides methods for adding new embeddings to the index.
  - Implements a `search` method to find the `top-k` most similar chunks to a given query embedding.
  - Handles saving and loading the FAISS index and chunk metadata.

7.5 `Qwen14BGenerator`

- **Purpose:** Acts as a wrapper for interacting with the Qwen-3 14B LLM via the LM Studio API.
- **Key Functionality:**
  - Configures the API client to connect to the local LM Studio endpoint.
  - Provides methods for sending chat completion requests to the LLM.
  - Manages LLM parameters (e.g., temperature, max tokens).
  - Implements robust error handling and failover logic for API communication.
  - Formats input and output consistent with the LM Studio API requirements.

7.6 `ImprovedRAGWithQwen14B`

- **Purpose:** Orchestrates the entire RAG process, from query reception to LLM prompt construction and response generation. This is the central intelligence module.
- **Key Functionality:**
  - Receives user queries.
  - Interacts with `EnhancedVectorStore` to retrieve relevant document chunks.
  - Implements the re-ranking logic (similarity + keyword bonus).

- Constructs the full LLM prompt, including system instructions, chat history, and retrieved context with inline citations.
- Calls `Qwen14BGenerator` to get the LLM's response.
- Parses the LLM's response to extract cited sources.
- Calculates the confidence score based on similarity, response length, and source diversity.
- Handles secure code generation prompt adjustments.

8. Usage Flow

The CyberSecure-Learn system is primarily interacted with via a Command Line Interface (CLI).

1. **Initialization:**
   - The user starts the application from the terminal.
   - The `ConfigManager` loads settings.
   - The `EnhancedVectorStore` loads the pre-built FAISS index and embedding model. If the index doesn't exist, the system prompts for PDF ingestion and index creation.
2. **PDF Corpus Ingestion (One-time or on update):**
   - The user specifies the directory containing cybersecurity PDF books.
   - `OptimizedPDFProcessor` reads each PDF.
   - `AdaptiveChunker` segments the extracted text.
   - `EnhancedVectorStore` generates embeddings and adds them to the FAISS index, which is then saved to disk.
3. **Starting a Chat Session:**
   - The `Chat Controller` loads the `conversation_history.json` file.
   - The user is prompted to select an existing session or start a new one.
4. **Interactive Tutoring Loop:**
   - **User Input:** The user types a question into the CLI.
   - **Query Processing:** The `Chat Controller` passes the query to `ImprovedRAGWithQwen14B`.
   - **RAG Process:** `ImprovedRAGWithQwen14B` performs semantic search, re-ranks chunks, builds the prompt, and sends it to `Qwen14BGenerator` (which interfaces with LM Studio).
   - **LLM Response:** Qwen-3 14B generates an answer based on the provided context.
   - **Output Display:** The `Chat Controller` receives the answer, confidence score, and attributed sources. It then formats and prints this information to the CLI.
   - **History Update:** The current turn (question, answer, sources, confidence, timestamp) is appended to the `conversation_history.json` file.
   - **Long Response Saving:** If the answer exceeds 1000 characters, it's saved to a timestamped text file.
   - The loop continues, allowing for a sustained conversation.

5. **Exiting:** The user can exit the application, and the `conversation_history.json` is ensured to be saved.

9. Technologies Used

Category	Technology/Library	Version (Example)	Role
Programming Language	Python	3.9+	Primary development language.
Local LLM	Qwen-3 14B	N/A	Core generative AI model for answers and code generation.
LLM Hosting	LM Studio API	Latest	Local server for hosting and serving Qwen-3 14B.
Vector Database	FAISS (Facebook AI Similarity Search)	`faiss-cpu`	Efficient similarity search on large vector datasets.
Embeddings	SentenceTransformer	`all-MiniLM-L6-v2`	Converts text chunks into numerical vector representations.
PDF Processing	PyMuPDF (`fitz`)	Latest	Extracts text and metadata from PDF documents.
Text Processing	NLTK	Latest	Used for sentence tokenization in adaptive chunking.
HTTP Requests	`requests`	Latest	For interacting with the LM Studio API.
AI API Client	`openai` (compatible)	Latest	API client for LM Studio, offering `completion` and `chat` endpoints.
Numeric Computing	`torch`	Latest	Underlying dependency for SentenceTransformers.
CLI Argument Parsing	`argparse`	Standard	Handles command-line arguments for bot interaction.
Data Serialization	`json`	Standard	Stores chat history persistently.

10. Known Limitations

Despite its robust features, the current MVP of CyberSecure-Learn has certain limitations:

- **No Graphical User Interface (UI):** The system operates solely via a Command Line Interface (CLI), which may limit accessibility for non-technical users.
- **Local-Only Operation:** While a core design principle for privacy, it currently precludes cloud-based collaboration or remote access.
- **Single-User Focus:** The current architecture is designed for a single local user; multi-user support or concurrent sessions are not implemented.
- **Dependency on LM Studio:** Requires the user to have LM Studio installed and running Qwen-3 14B locally, adding an installation prerequisite.
- **No Real-time PDF Updates:** Adding new PDFs requires manual re-indexing of the corpus.
- **Confidence Score Heuristic:** The confidence score is based on a heuristic combining similarity, length, and source diversity, and may not perfectly reflect actual factual accuracy or completeness.

- **Limited Ethical Filtering:** While a system prompt is in place, the depth of ethical filtering and safety guardrails is dependent on the LLM's inherent capabilities and may not prevent all undesirable outputs.

11. Future Roadmap

The future development of CyberSecure-Learn will focus on enhancing its capabilities, usability, and scalability.

- **RAG Tuning and Optimization:**
  - Experiment with advanced re-ranking algorithms (e.g., reciprocal rank fusion, cohort re-ranking).
  - Investigate multi-stage retrieval for complex queries.
  - Optimize chunking strategies for different document types.
  - Explore fine-tuning of the embedding model for cybersecurity-specific vocabulary.
- **LLM Fine-tuning (QLoRA):**
  - Explore QLoRA (Quantized Low-Rank Adaptation) for fine-tuning Qwen-3 14B on a smaller, highly curated cybersecurity dataset to improve domain-specific knowledge and response quality without requiring massive computational resources.
- **User Interface (UI) Integration:**
  - Develop a web-based or desktop graphical user interface for a more intuitive and user-friendly experience. This would likely involve frameworks like Flask/Django for backend and React/Vue.js for frontend.
- **Multi-user Support:**
  - Implement mechanisms for multiple users to interact with the system, potentially through a networked backend.
- **Enhanced Ethical AI and Safety:**
  - Integrate more sophisticated ethical filtering layers, potentially using external safety models or stricter prompt engineering.
  - Implement proactive content moderation.
- **Dynamic Corpus Updates:**
  - Develop features for seamless, real-time addition and indexing of new PDF documents without requiring a full re-ingestion.
- **Advanced Evaluation Metrics:**
  - Introduce more rigorous quantitative and qualitative evaluation metrics for RAG performance and LLM answer quality, potentially involving human in the loop.
- **Interactive Secure Coding Sandbox:**
  - Integrate a lightweight, in-app code execution environment to allow users to experiment with generated secure code snippets.

Appendix: Sample Query + Response Output

The following is a redacted example of a user interaction with CyberSecure-Learn, demonstrating the input, generated response, confidence score, and source attribution.

```
[User]: What are common vulnerabilities in web applications and how can they be prevented?

<Thinking... Retrieving relevant context...>

[CyberSecure-Learn]: Common vulnerabilities in web applications often stem from insecure coding practices and misconfigurations, allowing attackers to compromise data integrity, confidentiality, and availability. Key vulnerabilities include:

1.  **Injection Flaws:** Such as SQL Injection, NoSQL Injection, OS command injection. These occur when untrusted data is sent to an interpreter as part of a command or query. Prevention involves using prepared statements with parameterized queries, input validation, and escaping user-supplied data [Source 1: OWASP Top 10 Guide.pdf - Chunk ID: 15].
2.  **Broken Authentication:** Weak authentication or session management allows attackers to compromise passwords, keys, or session tokens. Robust prevention includes multi-factor authentication (MFA), strong password policies, secure session management (e.g., rotating session IDs), and server-side session timeouts [Source 2: Web Application Security.pdf - Chunk ID: 88].
3.  **Cross-Site Scripting (XSS):** This enables attackers to inject client-side scripts into web pages viewed by other users. Prevention requires proper input validation, output encoding, and using Content Security Policy (CSP) headers to restrict script execution [Source 1: OWASP Top 10 Guide.pdf - Chunk ID: 22].
4.  **Insecure Deserialization:** Occurs when untrusted data is used to deserialize objects, leading to remote code execution or denial-of-service. Preventing this involves avoiding deserialization of untrusted data and implementing integrity checks on serialized objects [Source 3: Advanced Web Security.pdf - Chunk ID: 45].
5.  **Security Misconfiguration:** Commonly seen due to insecure default configurations, incomplete configurations, open cloud storage, or misconfigured HTTP headers. Prevention includes hardening servers, disabling unnecessary features, patching systems regularly, and implementing automated scanning for misconfigurations [Source 2: Web Application Security.pdf - Chunk ID: 105].

By understanding these vulnerabilities and implementing the corresponding preventive measures, developers can significantly enhance the security posture of their web applications.

Confidence: 0.89

📚 Sources:
1. OWASP Top 10 - 2021: A Guide for Developers.pdf (Chunk 15, Score: 0.92)
2. Web Application Security: A Developer's Guide.pdf (Chunk 88, Score: 0.85)
3. Advanced Web Security: Attacks and Defense.pdf (Chunk 45, Score: 0.81)
4. OWASP Top 10 - 2021: A Guide for Developers.pdf (Chunk 22, Score: 0.79)
5. Web Application Security: A Developer's Guide.pdf (Chunk 105, Score: 0.77)
```