**Architecture Document for Software Dependency Risk Forecasting System**

Effective dependency management is essential for ensuring software security and system resilience. The intricate network of software packages creates a ripple effect, where vulnerabilities in a single dependency can propagate across the entire ecosystem, amplifying risk and complicating remediation efforts. As software projects grow in complexity, understanding the cumulative impact of vulnerable dependencies becomes increasingly challenging. This project proposes the development of a Software Dependency Forecasting System specifically tailored for Rust projects. The system processes a dataset of Rust dependencies, constructs a directed graph, and calculates a vulnerability score for each component. It models the relationships between crates, tracking how vulnerabilities propagate through dependency chains. The scoring mechanism utilizes a custom formula that assigns elevated scores to crates with highly vulnerable upstream connections, effectively quantifying the overall risk contribution of each element. By visualizing the dependency graph and dynamically updating scores as new vulnerabilities are discovered, the system provides actionable insights. This empowers Rust development and security teams to systematically identify and mitigate critical vulnerabilities, prioritize remediation efforts based on risk severity, and proactively strengthen the security posture of their software systems. The proposed approach enhances traditional vulnerability management practices, providing a more holistic view of software risk in evolving Rust ecosystems.

**Dataset Collection:**

* Gathers Rust crate data and Vulnerability dataset from source.
* Parses and preprocesses the data to identify dependencies and known vulnerabilities.

**Graph Construction Module:**

* Builds a directed graph where nodes represent crates and edges represent dependency links.
* Stores initial vulnerability scores as node attributes.

**User Interface:**

* Allows users to input a Rust project for analysis.
* Triggers the graph analysis process.

**Graph Traversal Module:**

* Uses BFS or DFS to explore connected dependencies.
* Collects vulnerability scores of upstream dependencies.

**Scoring Engine:**

* Applies a custom formula to calculate the overall vulnerability score.
* Aggregates upstream scores.

A diagram of a software development process

AI-generated content may be incorrect.

*Figure 1 Architecture Diagram*

**Result:**

* Classifies the crate as Low, Medium, or High Risk based on score thresholds.
* Shows the final vulnerability score and risk level.

The system starts with **Dataset Collection**, where data on Rust crate dependencies and vulnerabilities is gathered from sources like crates.io and RustSec. This data feeds into the **Graph Construction** module, which builds a directed graph with nodes as dependencies and edges as their links, storing initial vulnerability scores.

When a user inputs a dependency through the **User Interface**, the system triggers **Graph Traversal** to find the input dependency and explore its connected nodes. The **Scoring Engine** then applies a custom formula, aggregating the upstream vulnerability scores to compute a final risk score.

The **Results** module displays the calculated score and categorizes the risk as **Low**, **Medium**, or **High**, empowering developers to make informed security decisions. The feedback loop ensures that users can quickly assess and address vulnerabilities, strengthening their project’s security posture.