Project Title: Predictive Exploitability Modeling for Emerging Vulnerabilities

---

Core Idea:

- When new CVEs appear, **predict whether they will be weaponized soon** (i.e., will an exploit exist?) and prioritize patches based on **future risk**.

Component	Role
Threat Intelligence Collector	Monitors CVE/NVD feeds, social media, deep/dark web chatter
Exploit Prediction LLM	Predicts which new vulnerabilities will be exploited soon
Impact Severity Analyzer	Calculates potential real-world consequences
Exploitability Forecast Engine	Combines likelihood and severity into urgency scores
Forecast Reporting Module	Creates early warning bulletins for patching decisions

---

Component Details:

1. **Threat Intelligence Collector**:
    - Ingests:
        - CVEs (NVD, vendor advisories, etc)
        - Hacker forums
        - Exploit trading sites (EDB, etc)
        - GitHub PoC uploads
        - Etc
2. **Exploit Prediction LLM**:
    - Analyzes:
        - Historical CVE exploitation patterns.
        - CVE complexity, popularity, availability of PoCs, etc.
    - Outputs probability of near-term exploitation.
3. **Impact Severity Analyzer**:
    - Assesses:
        - Privilege level gain (e.g., RCE vs DoS, etc).
        - Affected asset criticality.
4. **Exploitability Forecast Engine**:
    - Combines exploit likelihood + impact into urgency score.
5. **Forecast Reporting Module**:
    - Provides **future risk forecasts**:
        - "Patch CVE-XXXX-YYYY in 72h; likely weaponized soon."

Overall System Flow:

- Input: New vulnerabilities appearing daily
- Output: Forecasted urgent patching priorities
- Focus: **Stay ahead of exploit development curve**

Internal Functioning of Each Module:

1. Threat Intelligence Collector

- **Sources**:
  - CVE/NVD feeds (e.g., via NIST API).
  - Threat feeds (RecordedFuture, VirusTotal Intel, etc).
  - Dark web forums (for early exploit chatter).
  - EDB.
  - Etc.

---

2. Exploit Prediction LLM

- **Inputs**:
  - CVE content (vulnerability summary, CVSS metrics).
- **Reasoning**:
  - LLM prompts:
    - `"Given the CVSS vector AV:N/AC:L/PR:N/UI:N and popularity of affected software, predict if public PoC will emerge within 30 days."`
- **Features considered**:
  - Attack complexity.
  - Popularity of the target software.
  - Historical behavior of similar vulnerabilities.

---

3. Impact Severity Analyzer

- **Severity modeling**:
  - Privilege escalation levels.
  - Critical asset exposure if exploited.
  - Etc.

---

4. Exploitability Forecast Engine

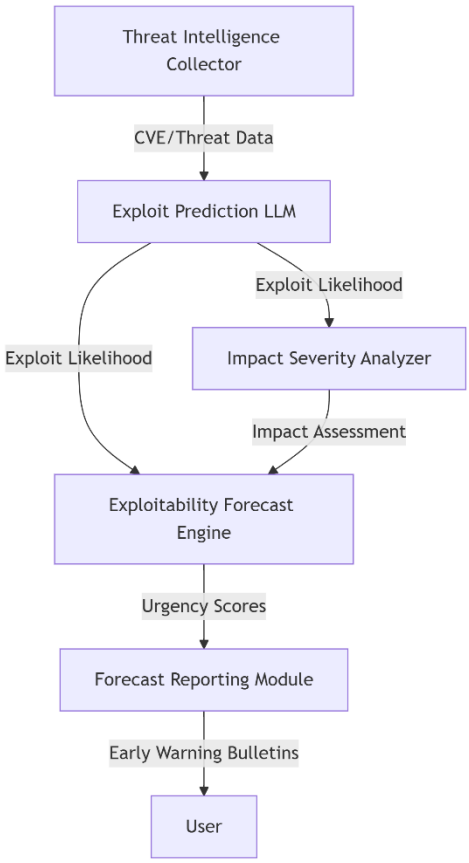
- **Forecasting Model**:
  - Likelihood × Impact → Urgency index.

---

5. Forecast Reporting Module

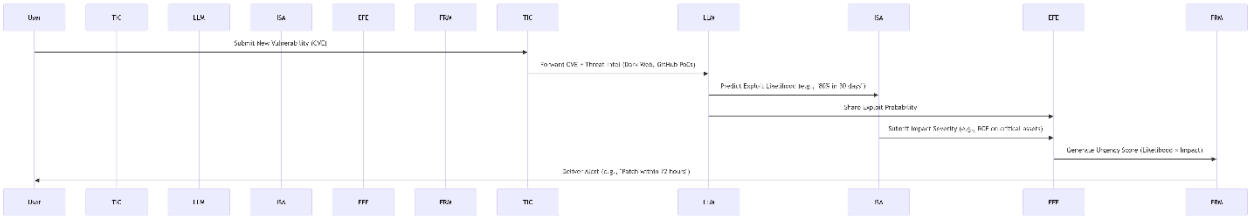
- **Alerts**:
  - CVEs with **high predicted weaponization risk**.
  - Recommended patch deadlines (e.g., **"patch within 72 hours"**).

Component Diagram



1. **Threat Intelligence Collector**: Aggregates data from CVE feeds, dark web forums, exploit repositories (e.g., GitHub PoCs), EDB, etc.
2. **Exploit Prediction LLM**: Predicts the likelihood of a CVE being weaponized based on historical patterns and software popularity.
3. **Impact Severity Analyzer**: Evaluates potential damage (e.g., remote code execution vs DoS, etc) and asset criticality.
4. **Exploitability Forecast Engine**: Combines exploit likelihood and impact severity into urgency scores.
5. **Forecast Reporting Module**: Generates actionable alerts (e.g., "Patch CVE-XXXX-YYYY immediately")

Sequence Diagram



1. **User** submits a new CVE to the **Threat Intelligence Collector**.
2. **Collector** gathers related threat intel (e.g., dark web chatter, etc) and sends it to the **Exploit Prediction LLM**.
3. **LLM** predicts exploit likelihood (e.g., "80% chance of PoC within 30 days") and shares results with the **Impact Severity Analyzer** and **Forecast Engine**.
4. **Severity Analyzer** assesses impact (e.g., "RCE on a public-facing server") and sends it to the **Forecast Engine**.
5. **Forecast Engine** calculates urgency scores (e.g., 9.5/10) and triggers the **Reporting Module**.
6. **Reporting Module** delivers prioritized alerts to the **User** (e.g., "Critical: Patch within 72 hours").

Detailed Project Description: Predictive Exploitability Modeling for Emerging Vulnerabilities

A framework that predicts the weaponization likelihood of new vulnerabilities (CVEs) and prioritizes patches based on future risk. This framework combines threat intelligence, Large Language Models (LLMs), and impact analysis to enable proactive defense.

---

1. System Architecture Overview

Core Components & Interactions

1. **Threat Intelligence Collector**
    - *Inputs*: CVE/NVD feeds, dark web forums, GitHub PoCs, exploit marketplaces, EDB, etc.
    - *Outputs*: Aggregated vulnerability and exploit-related data.
2. **Exploit Prediction LLM**
    - *Inputs*: CVE metadata, historical exploit patterns.
    - *Outputs*: Weaponization likelihood (e.g., "80% chance of PoC within 30 days").
3. **Impact Severity Analyzer**
    - *Inputs*: CVE details, asset criticality.
    - *Outputs*: Severity scores (e.g., "RCE on public-facing server").
4. **Exploitability Forecast Engine**
    - *Inputs*: Likelihood + severity data.
    - *Outputs*: Urgency scores (e.g., 9.5/10).
5. **Forecast Reporting Module**
    - *Inputs*: Urgency scores.
    - *Outputs*: Actionable alerts (e.g., "Patch within 72 hours").

Integration Flow:

1. Threat Intelligence → Exploit Prediction LLM → Impact Analyzer → Forecast Engine → Reporting Module.
2. Feedback loop: Validate predictions against real-world exploit emergence.

---

2. Component Implementation Details

2.1 Threat Intelligence Collector

Objective: Aggregate data from diverse sources to detect early exploit signals.

Tools & Workflow:

- **Data Sources**:
  - *CVE/NVD Feeds*: Use NIST API for real-time CVE updates.
  - *Dark Web*: Scrape forums (e.g., Dread, Exploit.in) using Tor and `scrapy` with CAPTCHA bypass.
  - *GitHub*: Monitor PoC repositories with GitHub API and `github3.py`.
  - *EDB*
  - *Etc*
- **Example Code**:

```
import requests
from bs4 import BeautifulSoup

def scrape_darkweb(url):
    session = requests.session()
    session.proxies = {'http': 'socks5h://localhost:9050'}
    response = session.get(url)
    soup = BeautifulSoup(response.text, "html.parser")
    return soup.find_all("div", class_="post")
```

- **Validation**: Deduplicate entries and flag false positives (e.g., fake exploit listings).

2.2 Exploit Prediction LLM

Objective: Predict exploit emergence using LLM analysis of CVE metadata and trends.

Implementation Steps (e.g.):

1. **LLM Setup**:

- *Base Model*: GPT-4 or fine-tuned Llama 2 on historical CVE-exploit pairs (e.g., CVE-2021-44228 Log4j), etc.
- *Prompt Engineering*:

```
prompt = f"""
CVE: {cve_description}
CVSS: {cvss_vector}
Affected Software: {software_name} (Downloads: 10M/month)
Predict the likelihood of a public PoC within 30 days.
"""
response = openai.ChatCompletion.create(model="gpt-4", messages=[{"role": "user", "content": prompt}])
# Output: "High (80%) due to widespread software usage and low attack complexity."
```

2. **Feature Extraction**:
   - *Software Popularity*: Pull download stats from npm, PyPI, or Docker Hub.
   - *Attack Complexity*: Parse CVSS vector metrics (e.g., `AC:L`).
3. **Validation**:
   - Compare predictions with historical data (e.g., "Log4j PoC emerged in 2 days").

2.3 Impact Severity Analyzer

Objective: Quantify potential damage from exploitation.

Implementation Strategies (e.g.):

1. **Severity Factors**:
   - *Privilege Escalation*: Classify as "RCE," "DoS," or "Information Disclosure."
   - *Asset Criticality*: Pull from CMDB (e.g., "public-facing server" = critical).
2. **Scoring Model**:

```
def calculate_severity(privilege_level, asset_criticality):
    severity_scores = {"RCE": 10, "DoS": 6, "InfoLeak": 4}
    return severity_scores[privilege_level] * asset_criticality
```

3. **Integration**:
   - Link to asset databases (ServiceNow CMDB) for real-time criticality updates.

2.4 Exploitability Forecast Engine

Objective: Generate urgency scores combining likelihood and impact.

Implementation Steps (e.g.):

1. **Scoring Formula**:
   ```
   Urgency Score = (Likelihood × 0.7) + (Severity × 0.3)
   ```
2. **Thresholds**:
   - *High Risk*: ≥8.0 → Patch within 72h.
   - *Medium Risk*: 5.0–7.9 → Patch within 14 days.
3. **Example Code**:
   ```
   def calculate_urgency(likelihood, severity):
       return (likelihood * 0.7) + (severity * 0.3)
   ```

2.5 Forecast Reporting Module

Objective: Deliver prioritized alerts to stakeholders.

Implementation Details (e.g.):

- **Alert Types**:
  - *Email Alerts*: Send via SMTP with severity labels (Critical/High/Medium).
  - *Ticketing*: Auto-create Jira tickets with deadlines.
- **Dashboard**:
  - *Tools*: Grafana or Elastic Kibana for real-time CVE tracking.
  - *Metrics*: Top CVEs by urgency, patch compliance rates, etc.
- **Example Output [markdown]**:
  ```
  **Alert: Critical**
  - **CVE**: CVE-2023-XXXX
  - **Urgency**: 9.5/10
  - **Action**: Patch within 72h.
  ```

```
- **Reason**: RCE on public-facing payment gateway (PoC expected in 7 days
).
```

---

3. Evaluation Metrics

1. **Prediction Accuracy**:
    - Precision/recall for exploit emergence within forecasted windows.
2. **Patch Timeliness**:
    - Reduction in time-to-patch high-risk CVEs vs. traditional methods.
3. **False Positives**:
    - Percentage of non-weaponized (exploited) CVEs flagged as high risk.

---

4. Challenges & Mitigation (optional)

- **Data Noise**:
    - Use NLP to filter irrelevant dark web chatter (e.g., `spaCy` for entity recognition).
- **LLM Hallucinations**:
    - Validate predictions against exploit DBs (Exploit-DB, Metasploit, etc).
- **Scalability**:
    - Use distributed scraping (Scrapy Cluster) and parallel LLM inference.

---

5. Tools & Technologies (e.g.)

- **LLM Framework**: Hugging Face Transformers, OpenAI API, etc.
- **Scraping**: Scrapy, Tor, `github3.py, etc`.
- **APIs**: FastAPI, NVD API, etc.
- **Database**: PostgreSQL, Redis, etc.
