PS1. Bot Profile Detection on Social Media

Challenge Overview

Social media platforms are increasingly affected by bot accounts that spread misinformation, spam content, and manipulate public discussions. These bots generate AI-powered content, making them harder to distinguish from real users. Traditional detection methods, such as rule-based filters, are often ineffective against sophisticated bots that randomize posting patterns and replicate human-like behavior.

This challenge focuses on developing a **bot detection system based on social media content and user activity**. Participants will analyze text features, posting frequency, and engagement metrics to identify automated accounts. The system should be scalable, efficient, and adaptable to various types of social media bot behaviors.

Objectives

1. **Bot Detection Mechanism**
    - Build a model that detects bots based on content patterns, sentiment, and linguistic features.
    - Identify anomalies in user activity, such as excessive automation, repeated content, and unnatural engagement levels.
    - Provide a confidence score indicating the likelihood of an account being a bot.
2. **Scalability and Performance**
    - Ensure the system can efficiently analyze thousands of social media posts in real time.
    - Design the pipeline to support large-scale data processing.
3. **Detection Reports and Insights**
    - Develop a structured output summarizing detection results.
    - Generate insights on bot activity trends and behaviors.

Evaluation Criteria

1. **Detection Accuracy**
    - Performance metrics such as precision, recall, F1 score, and AUC-ROC.
    - Ability to correctly identify AI-generated and spam content.
2. **Scalability and Performance**
    - Efficient processing of large volumes of social media data.
    - Real-time detection capabilities with minimal delay.
3. **Interpretability of Results**
    - Clarity in bot classification reports.
    - Effectiveness in distinguishing human-like bots from real users.

Implementation Guidelines and Suggestions

General Implementation Rules

- Core functionalities must be implemented from scratch.
- Copying code from platforms like Kaggle, GitHub, or similar is strictly prohibited.
- Participants must document their implementation process, including architecture, tools, and models used.

Dataset Usage

- Teams can use publicly available datasets focusing on bot detection using social media content, such as:
    - Twitter Bot Detection Dataset
- Feature extraction should focus on:
    - Text-based features (word embeddings, sentiment analysis, TF-IDF).
    - Posting patterns (time intervals, volume of posts, hashtag usage).
    - Engagement metrics (likes, shares, user mentions).

Advanced Models and Techniques

1. **Bot Detection Models**
    - **NLP Techniques**
        - Use transformers like BERT or RoBERTa for classifying social media content.
        - Apply TF-IDF, n-grams, or word embeddings to detect spam-like content.
    - **Behavioral Analysis**
        - Use anomaly detection models (Isolation Forest, Autoencoders) to identify unusual posting behaviors.
        - Implement time-series models (LSTMs) to track activity patterns over time.

    - **Ensemble Learning**
        - Combine ML and rule-based approaches for improved detection.
2. **Scalability Considerations**
    - Implement parallel processing with Apache Spark or Dask for large-scale data analysis.
    - Use optimized data pipelines for efficient processing.
3. **Privacy and Security**
    - Anonymize personally identifiable information (PII) in datasets.
    - Use encryption for data storage and transmission.

Deliverables

1. **Working Prototype**
    - A functional implementation that includes:
        - A bot detection mechanism based on social media content and user activity.
        - A structured output summarizing bot classification results.
    - Seamless integration of detection and reporting components.
2. **Cloud Deployment (Highly Encouraged)**
    - If possible, deploy the solution on a cloud platform.
    - Provide access credentials or a live URL for evaluation.
3. **Source Code**
    - Submit complete, modular, and well-commented source code.
    - Upload the code to a repository (GitHub, GitLab) and share access.
4. **Documentation**
    - Technical documentation explaining the architecture, models used, and tools.
    - Setup guide with instructions for local and cloud deployment.
    - Privacy measures implemented in the solution.
5. **Demo Video**
    - A 5–10 minute video showcasing:
        - The bot detection mechanism in action.
        - Insights from detection results.
        - Cloud deployment demonstration (if applicable).
6. **Evaluation Metrics**
    - Provide detection accuracy metrics (precision, recall, F1 score, AUC-ROC).
    - Summary of false positive/negative rates and model improvements.

Submission Format

- **Platform:** Submissions should be made via college emails.
- **File Structure:**
    - A compressed .zip file or repository link containing:
        - Source code folder.
        - Documentation files (README.md, TechnicalDocumentation.pdf).
        - Dataset or processed data (if applicable).
        - Demo video link (YouTube, Vimeo, or Google Drive).

PS2. AI-Powered Phishing Detection and Mitigation

Challenge Overview

Phishing campaigns are becoming more sophisticated, targeting individuals and organizations with personalized and deceptive content. Attackers use **spear-phishing emails** and **fake websites** to steal credentials, distribute malware, and gain unauthorized access. Traditional spam filters and blacklist-based defenses often fail, making AI-based detection crucial. This challenge focuses on **detecting and mitigating phishing attacks** through machine learning models and real-time threat intelligence.

Objectives

1. **Phishing Detection Mechanism:**
    - Develop a system to detect phishing emails and malicious websites.
    - Analyze email content, sender reputation, and embedded URLs.
    - Apply advanced content and URL analysis for identifying fake websites.
2. **Real-Time Threat Intelligence Integration:**
    - Leverage phishing indicators from sources like threat intelligence feeds and domain reputation databases.
    - Implement automated updates to enhance detection accuracy.
3. **Scalability and Performance:**
    - Ensure efficient processing of large volumes of emails and websites.
    - Support both real-time and batch-based analysis.
4. **Seamless Integration:**
    - Design the system to integrate with email security tools and web security platforms.

Evaluation Criteria

1. **Detection Accuracy:**
    - Performance metrics such as precision, recall, F1 score, and AUC-ROC.
    - Ability to detect sophisticated spear-phishing attacks and AI-generated phishing content.
2. **Speed and Scalability:**
    - Efficient processing of email traffic and website scans.
    - Real-time phishing detection with minimal latency.
3. **Integration and Deployment:**
    - Compatibility with email security tools, SIEMs, and web security platforms.
    - Ease of deployment and maintenance.

Implementation Guidelines and Suggestions

General Implementation Rules:

- Core functionalities must be implemented from scratch.
- Copying code from platforms like Kaggle, GitHub, or similar is strictly prohibited.
- Participants must document their implementation process, including architecture, tools, and algorithms used.

Dataset Usage:

- Teams can use publicly available phishing datasets, such as:
  - Phishing Database (for known malicious urls)
  - Enron Email Dataset (for training spear-phishing detection models)
- Feature extraction should include:
  - Email attributes (sender domain, link presence, attachment analysis)
  - Website characteristics (SSL certificate, domain age, WHOIS info)
  - Text-based features (NLP analysis of email body and subject line)

Advanced Models and Techniques:

1. **Phishing Detection Models:**
   - **NLP Techniques:**
     - Use transformers like BERT or RoBERTa to detect phishing email patterns.
     - Apply word embeddings and attention mechanisms to analyze email content.
   - **URL and Domain Analysis:**
     - Use random forest, gradient boosting (XGBoost), or deep learning for URL classification.
     - Implement graph-based models to analyze domain reputation and link structures.
   - **Website Screenshot Analysis:**
     - Apply CNNs (Convolutional Neural Networks) to detect visual similarities between phishing and legitimate sites.
   - **Ensemble Learning:**
     - Combine rule-based, ML, and deep learning methods for better detection.

2. **Real-Time Threat Intelligence:**

- Integrate with open-source threat feeds like OpenPhish, Abuse.ch, and VirusTotal APIs.
- Implement anomaly detection on email traffic patterns using Autoencoders or Isolation Forest.
- Use Kafka or RabbitMQ for streaming phishing alerts.

3. **Scalability Considerations:**
   - Implement distributed processing with Apache Spark or Dask for large-scale email analysis.
   - Containerize the application using Docker and Kubernetes for deployment.
4. **Privacy and Security:**
   - Encrypt all sensitive user data using AES-256.
   - Anonymize email content and sender details before processing.

Deliverables

1. **Working Prototype:**
   - A functional implementation, including:
     - Phishing detection mechanism for email and web threats.
     - Threat intelligence integration for real-time phishing alerts.
     - Dashboard for visualizing phishing reports.
   - Seamless integration of detection, visualization, and intelligence components.
2. **Cloud Deployment (Highly Encouraged):**
   - If possible, deploy the solution on a cloud platform.
   - Provide access credentials or a live URL for evaluation.
3. **Source Code:**
   - Submit complete, modular, and well-commented source code.
   - Upload the code to a repository (GitHub, GitLab) and share access.
4. **Documentation:**
   - Technical documentation explaining the architecture, models used, and tools.
   - Setup guide with instructions for local and cloud deployment.
   - Details on implemented privacy measures.
5. **Demo Video:**
   - A 5–10 minute video showcasing:
     - The phishing detection mechanism in action.
     - Threat intelligence integration and real-time alerts.
     - Cloud deployment demonstration (if applicable).
6. **Evaluation Metrics:**
   - Provide detection accuracy metrics (precision, recall, F1 score, AUC-ROC).
   - Summary of false positive/negative rates and model improvements.

Submission Format

- **Platform:** Submissions should be made via college emails.
- **File Structure:**
    - A compressed .zip file or repository link containing:
        - Source code folder
        - Documentation files (README.md, TechnicalDocumentation.pdf)
        - Dataset or processed data (if applicable)
        - Demo video link (YouTube, Vimeo, or Google Drive)

PS3.Cross-Chain Asset Transfer

Description:

- Develop a cross-chain asset transfer system where assets can be locked on one blockchain and minted as a representation on another. Participants will deploy gateway smart contracts on two test nets and implement a front-end UI to interact with these contracts. The solution should ensure security, transparency, and proof-of-transfer mechanisms.

Notes:

- Only newly deployed contract allowed
- No 3rd party bridge integration allowed
- Can use 3rd party libraries like open zepplin etc. in contracts

Core Features & Requirements:

Lock & Mint Mechanism

- Users should be able to lock an asset (ERC-20 or ERC-721) on Chain A and receive a minted equivalent on Chain B.
- Unlocking the asset on Chain A should burn or lock the equivalent asset on Chain B.

Proof of Transfer

- Every successful transfer should generate a receipt (hash, timestamp, sender, receiver, and amount).
- Users should be able to view and verify these proofs in the UI.

Security Checks

- Implement replay protection, contract access controls, and reentrancy protection.
- Validate user input and prevent unauthorized minting.
- Ensure gas-efficient execution.

Working UI with Wallet Connectivity

- Users must be able to connect their wallets (MetaMask AND Wallet Connect, etc.).
- UI should display asset balances, pending transfers, and successful transaction logs.

Technical Guidelines:

Smart Contract Development (Solidity)

- Deploy contracts on two testnets (e.g., Amoy & Sepolia).
- Use event-based architecture for tracking transfers.
- Implement a relayer or an off-chain simulation for cross-chain communication (e.g., using oracles, signature verification, or Merkle proofs)

Frontend Development

- Create an interactive UI using React, Next.js, or Vue.
- Display live transfer status, receipts, and logs.
- Provide extra features like transfer history, asset filtering, notifications, and progress indicators.

Evaluation & Scoring Metrics

- Solidity implementation, gas optimization, security measures
- Smooth wallet connection & user experience
- Additional functionalities like history, filters, and notifications
- Protection against attacks like reentrancy, spoofing, and replay attacks
- Higher audit scores lead to better ranking
- Bridge Safety

Bonus Points:

- On-Chain Verification: Implement Merkle proofs or zk-SNARKs for added trust.
- Decentralized Relayers: Remove reliance on centralized bridges.
- Gas Optimization: Reduce transaction costs via efficient contract design.

Expected Deliverables:

- Smart Contracts Deployed and Verified Onchain Addresses add in README
- Live Demo UI URL [HOSTED]
- Demo Video link [LOOM]
- Source Code on [GITHUB] w link
- README with Setup Instructions