**PhishNet X – Advanced Phishing Detection System** is a Python-based application built with a graphical user interface (GUI) using Tkinter. Its primary aim is to detect phishing attempts by performing a multi-layered analysis of URLs and emails. The system combines several techniques including heuristic checks, SSL and DNS validations, content analysis, and integration with external threat intelligence sources (like VirusTotal and WHOIS) to calculate a risk score and provide a final verdict on potential phishing risks.

**Core Components and Architecture**

**1. User Interface (UI)**

* **Tkinter & ttk Widgets:**  
  The GUI is built using Tkinter and themed with ttk. It uses a Notebook widget to separate different functionalities into three main tabs:
  + **URL Analysis Tab:** Allows the user to input a URL and start its analysis.
  + **Email Analysis Tab:** Provides fields for entering the sender, subject, and body of an email for phishing evaluation.
  + **Results Tab:** Displays detailed technical analysis and threat intelligence (including VirusTotal, WHOIS, geolocation, and SSL details).
* **Design and Responsiveness:**  
  The code configures a modern look (custom colors, fonts, and padding) and uses threading to run time-intensive analysis tasks in the background. This ensures that the user interface remains responsive while processing the analysis.

**2. Detection Rules and Threat Intelligence**

* **Detection Rules:**  
  The method load\_detection\_rules initializes a set of rules defining suspicious keywords, top-level domains (TLDs) that are often abused, trusted domains, and various patterns (like homograph, combosquatting, bitsquatting, and common misspellings). These rules serve as a baseline for many of the heuristics used during URL and email analysis.
* **External APIs and Databases:**
  + **VirusTotal API:** Used to assess URL reputation by fetching analysis statistics (malicious, suspicious, harmless counts) and converting them into a risk score.
  + **WHOIS:** Retrieves domain registration details such as creation and expiration dates, registrar, and registrant information.
  + **GeoIP2 Database:** Looks up geolocation information based on the IP address of a domain.

This multi-source approach enhances the accuracy of phishing detection.

**URL Analysis Details**

When a URL is submitted, the system follows several steps:

**A. Input Validation**

* **validate\_url Method:**  
  This function ensures the URL contains a proper scheme (http/https) and a valid network location. If missing, it automatically adds a default scheme.

**B. Basic URL Pattern Checks**

* **check\_basic\_url\_patterns Method:**  
  The code uses Python’s urlparse and tldextract libraries to break down the URL and examines various suspicious characteristics:
  + **IP-based URLs:** Detects if the URL uses an IP address instead of a domain name.
  + **Suspicious TLDs:** Compares the TLD of the URL against a list of known high-risk TLDs.
  + **URL Length & Subdomain Count:** Flags URLs that are unusually long or have an excessive number of subdomains.
  + **Suspicious Keywords & Encoding Abuse:** Searches for phishing-related keywords and excessive use of URL encoding.
  + **Advanced Checks:**  
    It further inspects for issues such as homograph attacks (using mixed scripts or Punycode indicators), combosquatting (suspicious prefixes), bitsquatting, common misspellings, repeated characters, excessive dashes, and numeric affixes.  
    These checks cumulatively contribute to a risk score for the URL.

**C. Security Feature Analysis**

* **check\_security\_features Method:**  
  This function verifies if the URL is served over HTTPS. If not, it immediately raises a high-risk flag.
* **get\_ssl\_certificate Method:**  
  If HTTPS is used, the method retrieves SSL/TLS details using Python’s ssl and socket libraries. It extracts key certificate details (SSL version, issuer, expiration date, and cipher suite) and checks for validity or expiry.

**D. Web Content Analysis**

* **analyze\_web\_content Method:**  
  Using the requests library and BeautifulSoup, this method:
  + Fetches the webpage content.
  + Scans for login forms which might indicate phishing pages.
  + Inspects the <script> tags to identify insecure external sources.
  + Checks for mixed content on HTTPS pages (e.g., images loaded over HTTP).  
    Each finding contributes to the overall risk evaluation.

**E. Reputation and Technical Data Gathering**

* **VirusTotal Reputation Check (check\_url\_reputation):**  
  The system converts the URL into an identifier (using Base64 encoding) and queries VirusTotal. It either retrieves the latest analysis or submits the URL for scanning and polls for results. The score is derived from the number of malicious or suspicious detections.
* **DNS and HTTP Response Analysis (update\_technical\_info):**  
  It resolves the domain to an IP address, fetches DNS records and nameservers using dns.resolver, and retrieves HTTP headers (status, server type, content type).
* **Domain Information (get\_domain\_info):**  
  This function uses the WHOIS module to obtain registration dates, domain age, and assigns a reputation status based on whether the domain is trusted or if it has a suspicious TLD.

**Email Analysis Details**

When analyzing an email, the application focuses on several key aspects:

**A. Input Fields and Basic Checks**

* The Email Analysis tab collects three inputs: **Sender**, **Subject**, and **Body**. The system validates that all these fields are provided before starting the analysis.

**B. Sender Analysis**

* **Domain Impersonation & Typosquatting:**  
  The code extracts the domain from the sender’s email address and compares it against known financial institutions (e.g., bankofamerica.com, chase.com). If the domain does not exactly match the legitimate domain (even if similar), it flags it as potentially impersonated.
* **Keyword-Based Checks:**  
  It uses regex to search for common suspicious words (like “support”, “security”, “urgent”) in the sender’s name. If such keywords are present and the domain isn’t in the trusted list, additional risk points are added.
* **Similarity Measurement:**  
  The function calculate\_domain\_similarity computes the Levenshtein distance between domains to help detect typosquatting.

**C. Subject Analysis**

* **Urgency and Action Requests:**  
  The subject line is scanned using predefined regex patterns that look for urgency (e.g., “immediate”, “urgent”, “action required”) and phrases suggesting account verification or restriction. Each detected pattern increases the risk score.

**D. Body Content Analysis**

* **URL Extraction and Analysis:**  
  The method identifies URLs within the email body and performs similar checks as in the URL analysis (e.g., detecting deceptive domains related to financial institutions).
* **Threat Language Detection:**  
  It searches for patterns that suggest imminent threats or time pressure (e.g., “within an hour”, “immediate action needed”).
* **Sensitive Information Requests:**  
  Patterns that ask the user to verify or update personal details are also flagged.

**E. Risk Score and Verdict**

* The email analysis aggregates risk points from the sender, subject, and body checks. The risk score is normalized (with a maximum of 100) and mapped to a final verdict:
  + **High Risk:** Likely phishing.
  + **Medium Risk:** Suspicious.
  + **Low Risk:** Some concerns.
  + **Safe:** Likely legitimate.
* **Results Display:**  
  The update\_email\_results method then outputs these results in a scrollable text widget, using color codes (red for high risk, orange/yellow for medium/low, green for safe) to visually differentiate the risk levels.

**Supporting Utilities and Error Handling**

**Utility Functions**

* **Domain Age and Typosquatting Checks:**  
  Functions such as is\_new\_domain and is\_typosquatting (using the Levenshtein distance algorithm) help determine if a domain is recently registered or suspiciously similar to well-known domains.
* **WHOIS Information Analysis:**  
  The method get\_whois\_info retrieves WHOIS data and flags risk factors like very recent domain creation, privacy-protected records, or missing information.
* **Geolocation Lookup:**  
  The get\_geoip\_info function uses the GeoLite2 database to provide additional context (country, city, coordinates, timezone) for the domain’s IP address.

**Error Handling**

* Throughout the code, try/except blocks are used to catch errors gracefully. Whether it’s resolving a domain name, connecting via SSL, or fetching external data, the code logs errors and provides user feedback (via status bars or message boxes) without crashing the application.

**Concurrency and UI Responsiveness**

* **Threading:**  
  Both URL and email analyses run in separate threads. This design choice prevents the GUI from freezing during potentially long-running network operations or data processing.
* **Dynamic UI Updates:**  
  The application continuously updates UI elements (like risk meters, status bars, and detailed result panels) using Tkinter’s thread-safe after method, ensuring that the analysis progress is visible to the user.

**Summary**

PhishNet X is a robust and multi-faceted phishing detection system. It:

* **Analyzes URLs** by validating their structure, inspecting DNS and SSL details, checking for suspicious patterns (both basic and advanced), evaluating webpage content, and integrating threat intelligence from VirusTotal and WHOIS.
* **Evaluates Emails** by dissecting sender details, subject lines, and email bodies to detect urgent language, impersonation attempts, and dangerous URLs.
* **Presents Findings** through a well-organized GUI that breaks down technical details and threat intelligence, providing clear risk scores and final verdicts.
* **Utilizes External Data Sources** and custom detection rules to provide a comprehensive risk assessment while maintaining high responsiveness through multi-threading and careful error handling.