**NATproxy**

By: Mohammad Zbeeb  
 Mohamad Bazzi  
 Mariam Salman  
 Mohammad Ghorayeb

*Institutional Affiliation*

EECE 351 - Project Report

**Project Report**

Professor: Ayman Tajeddine

Group: 13

Fall 2023/2024

**1. Introduction:**

**1.1 Purpose of the Proxy Server**

The proxy server implemented in this project serves as a middleware facilitating clients' access to HTTP web pages. Key functionalities include HTTP and HTTPS request parsing, caching, URL blocking, and other security features.

**1.2 Overview of Design Choices**

The proxy server, developed using Python and Flask, is a dynamic solution that integrates core components such as HTTP and HTTPS request parsing, caching policies, URL blocking/filtering and other robust security measures. Notably, it incorporates machine learning models—Linear Regression for cache expiration prediction and a RandomForestClassifier for redirect handling. These models introduce predictive analytics and security enhancements, respectively, elevating the proxy server's capabilities to adapt to varying network conditions and identify potential security threats.

**2. Features and Properties:**

**2.1 Choice of Programming Language (Python)**

Python, along with the Flask framework, was chosen for its simplicity, readability, and the availability of relevant libraries for networking and machine learning tasks.

**2.2 Listening on a Port**

The proxy server is configured to listen on port 5000 for incoming connections, balancing ease of use and avoiding potential conflicts with existing services.

**2.3 Parsing HTTP and HTTPS Requests**

HTTP/HTTPS requests from clients are parsed to extract relevant information, such as method, URL, headers, etc. The server logs these details along with the IP address and exact time of the request.

**2.4 Forwarding Requests to Web Server**

Client requests are forwarded to the target web server, and the proxy server waits for the corresponding response. Detailed logging captures the exact time of the request and the response from the web server.

**2.5 Receiving and Relaying Responses**

Upon receiving the HTTP response from the web server, the proxy server relays it back to the requesting client. Logging is implemented to record the times of receiving and sending the response.

**2.6 Error Handling**

The proxy server handles errors from both the client and server sides, displaying informative messages and returning appropriate error responses. Error messages are logged, providing insights for debugging.

**2.7 Caching Policy**

The proxy server implements a caching mechanism based on a linear regression model (more on ML in section 6). The cache expiration time is dynamically determined, and caching decisions are made based on both the cache expiration time and the presence of cached responses. The server checks the cache before forwarding requests to the web server.

**2.8 Filtering**

Filtering and blocking mechanisms are implemented to control user access to specific URLs. The code includes functionality to block users, block specific URLs for all users, block URLs for individual users, and allow or unblock users and URLs

**2.9 Security Features**

Security features include filtering based on URLs or IP addresses, denying internet access to specific clients, and basic user authentication. Filtering decisions are applied before processing client requests.

**3. Challenges Faced:**

**3.1 Model Training**

A critical challenge lies in guaranteeing that the model is trained with pertinent and current data. The current setup lacks a mechanism for continuous learning or periodic retraining. Introducing a system to update the model with new data over time would represent a substantial enhancement.

**3.2 Splitting the database:**

This challenge arises from the need to handle caching differently for HTTP and HTTPS requests (discussed in section 4.1). The challenge is to ensure that users can only access their own cached HTTPS responses while allowing shared access to cached HTTP responses. Designing the database schema to accommodate both global and user-specific caching wasn’t easy. We included fields that differentiate between HTTP and HTTPS responses and associate responses with specific users for HTTPS in the CachedResponse table.

**3.3 Eviction of Cache Items**

Managing the cache size and evicting items when the limit is reached required careful consideration to avoid unintended consequences. The use of an OrderedDict helps in efficiently managing the cache.

**4. Implementation Details:**

**4.1 HTTP/HTTPS Request Handling**

* **Protocol Detection:** The URL protocol (HTTP or HTTPS) is determined by parsing the requested URL.
* **Proxy View Function:** This function is responsible for handling incoming HTTP requests. It acts as a proxy, forwarding requests to a target server specified by the url. The requests\_args parameter allows customization of the request made to the target server.
* **Making Requests to the Target Server:** If no valid cached response is found or the cached response is expired, the code uses the requests library to make a new request to the target server.
* **Creating and Storing New Cached Response**: After receiving the response from the target server, a new CachedResponse object, including the HTTP method, URL, content, timestamp, and Last-Modified information, is created and stored in the database for potential future use based on th caching mechanism explained below.
* **Caching Mechanism**: **HTTP requests** are cached in a shared database accessible to all users of the proxy server. These responses are stored globally and are not specific to any individual user. The system retrieves and serves HTTP responses from this shared cache to enhance response time and reduce the load on the web servers. **HTTPS requests**, being sensitive and user-specific, are stored in individual caches tied to each authenticated user. Each user's HTTPS responses are maintained separately, ensuring privacy and access control. Only the authenticated user can access and retrieve their own cached HTTPS responses.
* **Handling Headers and Forwarding Response to Client:** The code extracts relevant headers from the original response received from the target server and includes them in the proxy response. Certain headers are excluded to prevent interference with the proxying process.
* **Adding Proxy Server Information to Headers:** The code adds information about the proxy server (e.g., hostname and protocol) to the headers of the proxy response. This information is commonly used in proxy scenarios to identify the intermediaries involved in request handling.
* **Returning Proxy Response to Client:** Returning the proxy response, either the cached response or the newly fetched response, to the client.

**4.2 Caching Capabilities**

* **Database Model:** The “CachedResponse” class defines the database model for storing cached responses. It includes fields such as method (HTTP method), url (requested URL), content (cached content), timestamp (time of caching), and last\_modified (Last-Modified header from the original response).
* **Checking cached response:** Before making a request to the target server, the code checks if there is a cached response for the given URL and HTTP method. If a cached response is found, the code proceeds to check for expiration based on the CACHE\_EXPIRATION\_TIME and 'Last-Modified' timestamp.
* **Checking Expiration:** If the cached response is still considered valid (not expired), it is returned to the client. The CACHE\_EXPIRATION\_TIME is a threshold that determines how long a cached response is considered fresh.

**4.3 Filtering**

* **Block User Functionality:** The code includes an admin route (/admin/block\_user) that allows an admin user to block a specific user. This is implemented in the block\_user route function. The admin user can send a POST request with the email of the user to be blocked. The targeted user's is\_blocked attribute is set to True in the database, preventing them from accessing the proxy server.
* **Block URL for All Users Functionality:** The admin can block a specific URL for all users by using the /admin/block\_url\_for\_all route. This route expects a POST request with the URL to be blocked. The code then iterates over all users and appends the blocked URL to their list of blocked URLs.
* **Block URL for Individual User Functionality:** There are two routes related to blocking URLs for individual users: /admin/block\_url\_for\_user and /admin/unblock\_url\_for\_user. The first route allows the admin to block a specific URL for a particular user, and the second route allows unblocking a URL for a user. Both routes expect a POST request with the user's email and the URL to be blocked or unblocked.
* **Handling Blocked URLs in the Proxy Functionality:** The proxy functionality in the proxy\_view route checks if the requested URL is in the blocked URLs list for the current user. If the URL is blocked, a JSON response with an error message is returned, preventing the proxy request.

**4.4 Security Features**

* **User Whitelisting:** Apply wide range of filters upon users
* **Domain Whitelisting:** Similar to IP whitelisting, a list of allowed domains (ALLOWED\_DOMAINS) is specified. Requests for domains not in this whitelist are restricted, providing another level of control over permitted traffic.
* **User Authentication and Authorization:** User authentication and session management are crucial security measures. Only authenticated users are allowed to access certain routes, such as the /index route.
* **Password Hashing:** When creating or updating user passwords, the code uses Flask-Bcrypt to securely hash passwords. Storing passwords as hashes enhances security by preventing exposure of plaintext passwords even if the database is compromised.

**4.5 Error Handling**

* **Global Exception Handling:** In the ‘proxy\_view’ function, a generic exception handler (requests.RequestException) is used to catch errors that may occur during the execution of the HTTP request to the target URL. If an exception is caught, it prints an error message and returns a generic response with a 500 Internal Server Error status.
* **Database Commit Error Handling:** When committing changes to the database, the code includes a try-except block to handle potential exceptions that might occur during the commit process. If an exception is raised, it prints an error message.
* **Specific Database Query Handling:** While not explicitly an error, the code checks if a cached response exists based on the query. If a cached response is not found, it prints a message indicating that no cached response was found. This is a form of handling the absence of expected data.
* **Exception Handling during Login:** In the login route, there is a try-except block that captures exceptions during the login process, including any errors that might occur during the interaction with the SMTP server to send a welcome email. If an exception is caught, it prints an error message.
* **HTTP Request Handling and Response Status:** The code checks for the expiration of cached responses and handles the HTTP response status accordingly. Depending on whether the cached response has expired or not, the code returns a response with an appropriate HTTP status code (200 for a valid cached response or 404 for an expired one).

**4.6 Logging**

* Comprehensive logging of request and response times.
* Logging of error messages for debugging and troubleshooting.

**5. Testing Overview:**

**5.1 Unit Testing**

Individual components of the proxy server were tested in isolation to ensure functionality.

**5.2 Integration Testing**

The proxy server was tested as a whole, ensuring all components work together seamlessly.

**5.3 Security Testing**

The effectiveness of implemented security features was tested to identify potential vulnerabilities.

**6. Additional Features:**

**6.1 User Interface**

The user interface (UI) of the application is designed to provide a seamless and visually appealing experience for users interacting with the proxy service:

* The UI includes a URL input form allowing users to submit URLs for proxy browsing.
* Options for social login with Google and Facebook enhance user convenience and accessibility.
* Responsiveness: The UI is designed to be responsive, ensuring a consistent and user-friendly experience across various screen sizes and devices.

**6.2 Machine Learning Integration**

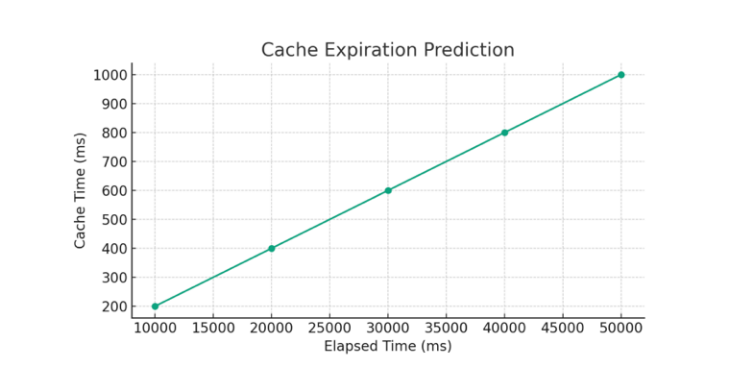
The server integrates machine learning models for cache expiration prediction and potential malicious redirect detection.

**Cache Expiration Prediction:**

A LinearRegression model is utilized to dynamically predict cache expiration times based on historical data. LinearRegression is selected due to its simplicity and effectiveness in modeling linear relationships. It is presumed that a linear correlation exists between request fetch time and the optimal cache duration. This process seeks to strike a balance between maintaining a fresh cache and reducing server load by minimizing frequent fetches from the internet.

Implementation Details:

* **Data Collection:** The dataset is formed using elapsed times (elapsed\_times) for fetching content from the internet and the corresponding times these items were retained in the cache (cache\_times).
* **Training Process:** The model is trained with elapsed\_times as the independent variable and cache\_times as the dependent variable. This training establishes a correlation between the time taken to fetch data and how long that data should be cached. The training occurs within the script, rendering the model ready for predictions after this initial training phase.
* **Usage in Application:** Whenever a new request is made, the model predicts the cache expiration time based on the request's fetch time. This dynamic approach enables the application to adapt to changing network conditions and content update frequencies.

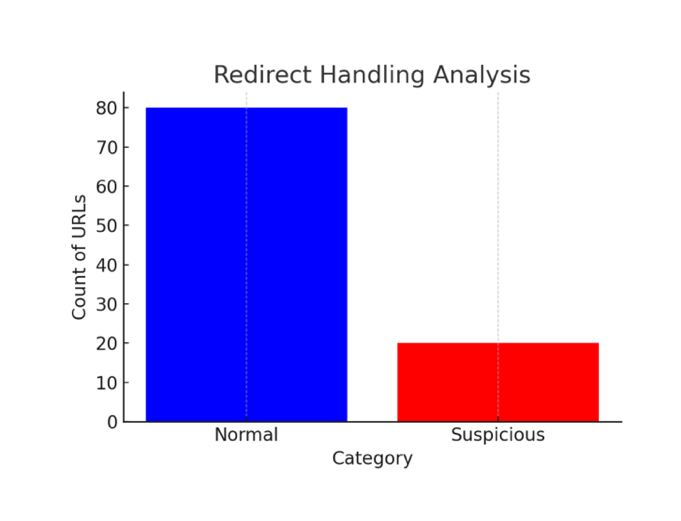


**Redirect Handling:**

The purpose is to identify potentially malicious redirects that could be indicative of phishing attacks or other security threats. A RandomForestClassifier is employed for its proficiency in handling categorical data and its robustness against overfitting.

Implementation Details:

* **Feature Extraction:** Features such as the domain and path length of URLs are extracted. These features are likely chosen based on the assumption that malicious redirects might exhibit specific patterns in domains or path structures.
* **Classification Process:** The model is presumed to be trained on a labeled dataset (not provided in the script) that categorizes URLs as normal or suspicious. In the application, for each redirect URL, these features are input into the model to classify the URL as normal or potentially malicious.
* **Alert System:** If a URL is classified as potentially malicious, an alert function (log\_alert) is triggered, enhancing the security aspect of the proxy server.



**6.3 Preventing DDOS Attacks**

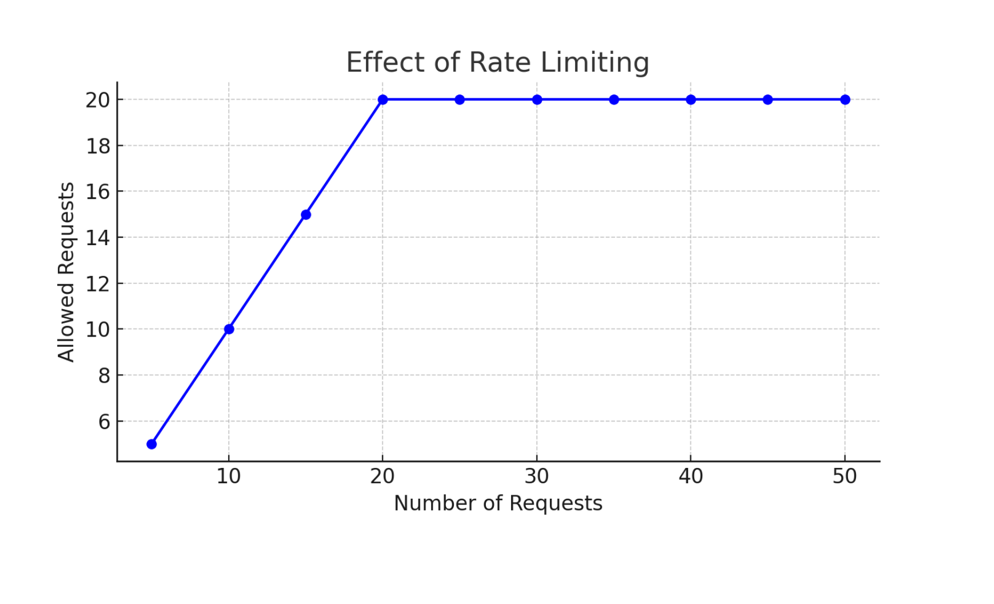
Flask Limiter serves as a robust defense against Distributed Denial of Service (DDoS) attacks by strategically controlling the rate of incoming requests to a Flask application. This not only prevents undue strain on resources but also fortifies the application against malicious attempts to overwhelm it.

**How Flask Limiter Works**

* Rate Limiting: Flask Limiter operates on the principle of rate limiting, enabling developers to regulate the frequency of requests directed at the Flask application. This serves as a proactive measure to curb excessive resource consumption.
* Configuration Options: Developers wield the flexibility to configure limits on a per-route basis, globally, or by considering IP addresses. Flask Limiter employs various storage backends to meticulously track request rates, providing a versatile and adaptable approach.

**DDoS Attack Prevention Mechanisms**

* Reducing Impact: By capping the number of requests a user can make within a specified time frame, Flask Limiter significantly diminishes the potential impact of DDoS attacks. This restriction curtails the attacker's ability to inundate the application with an overwhelming volume of requests.
* Early Detection: Flask Limiter acts as an early warning system. Any surge in requests surpassing the predefined limits triggers alerts, enabling swift identification of a possible DDoS attack. This proactive stance empowers administrators to take timely countermeasures.



**6.4 Email Notification**

A feature is implemented to send a welcome email to users upon successful login, showcasing additional functionality beyond the basic requirements. This adds a layer of user interaction and notification, ensuring that users are informed about successful logins and are aware of the application's features.

**6.5 Admin Interface:**

The admin user interface in the proxy server is designed for controlling and managing various aspects of the system. Here's an overview of its functionality:

* The admin panel provides several buttons, each associated with a specific action:
  + **Block this User:** Blocks the user associated with the provided email.
  + **Block URL For This User:** Blocks the specified URL for the user associated with the provided email.
  + **Block URL For all Users:** Blocks the specified URL for all users.
  + **Allow User Access:** Allows access for the user associated with the provided email.
  + **Unblock URL For This User:** Unblocks the specified URL for the user associated with the provided email.
  + **Unblock URL For all Users:** Unblocks the specified URL for all users.
  + **LOG OUT:** Logs out the admin user.
* **Security Measures:** Access to the admin panel is restricted to the admin user with the email ("[natproxyy@outlook.com](mailto:natproxyy@outlook.com)").
* **Email Notification:** Upon successful login, a welcome email is sent to the admin's email address ("[natproxyy@outlook.com](mailto:natproxyy@outlook.com)").
* **Integration with Flask Framework:** Built using the Flask web framework, incorporating components like Flask-SQLAlchemy, Flask-Bcrypt, and Flask-Login for user authentication.
* **Machine Learning Models:** The admin panel interacts with the machine learning models implemented in the proxy server, enhancing security features such as redirect handling and cache expiration prediction.

User Interface:

A computer screen shot of a person sitting at a desk

Description automatically generated

A screenshot of a computer

Description automatically generated

Admin Interface:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A white screen with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated