iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.80201

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

ExposeHost: Exposing Localhost to Internet

Yash Patil[1], Sairaj Pai[2], Nandini Nichite[3], Dr. Devika Rani Roy[4]
Information Technology, K.C. College of Engineering

Abstract: ***In an era dominated by cloud computing and the Internet of Things (IoT), the ability to access local services remotely is a fundamental requirement. However, the exhaustion of IPv4 addresses and the widespread implementation of Carrier-Grade NAT (CGNAT) by Internet Service Providers (ISPs) have made traditional port forwarding increasingly difficult and insecure. This paper presents "ExposeHost," a secure, self-hosted, end-to-end port forwarding solution designed to expose local TCP and HTTP services to the public internet. Unlike commercial solutions that function as "black boxes," ExposeHost offers a transparent, Python-based architecture leveraging asyncio for high-concurrency connection multiplexing. It introduces a custom JSON-header-based protocol over TLS for secure control channels and integrates a client-side authentication proxy (AuthProxy) using HMAC-signed cookies to secure otherwise unprotected local services. We evaluate ExposeHost against existing solutions like Ngrok and FRP (Fast Reverse Proxy), demonstrating its efficacy in providing a secure, performant, and customizable alternative for developers and self-hosters.***

Keywords: Port Forwarding, NAT Traversal, Tunneling, Reverse Proxy, Cloud Computing.

I. INTRODUCTION

The growth of the Internet has led to a critical shortage of IPv4 addresses, compelling ISPs to adopt Network Address Translation (NAT) and, more aggressively, Carrier-Grade NAT (CGNAT) [10]. While effective for conserving address space, these technologies break the end-to-end principle of the internet, making devices on a Local Area Network (LAN) inaccessible from the Wide Area Network (WAN) without complex configurations or third-party intermediaries.

Simultaneously, the rise of "self-hosting" and the need for developers to demonstrate local web applications to remote clients have created a demand for tunneling solutions. Traditional methods like VPNs are often too heavy for simple service exposure, while SSH remote port forwarding can be cumbersome to manage for multiple services.

II. PROBLEM STATEMENT

Existing commercial tunneling solutions, such as Ngrok [7][12], offer ease of use but often come with limitations on bandwidth, concurrent connections, and persistent domain names in their free tiers. Moreover, relying on a third-party service introduces privacy concerns, as all traffic is decrypted at the provider's edge before being re-encrypted. Open-source alternatives like FRP [20] are powerful but can be complex to configure and lack integrated, user-friendly authentication mechanisms for the exposed services.

A. Objectives

This paper proposes ExposeHost, a system designed to:

1) Bypass NAT/CGNAT: Enable reliable inbound connectivity to local devices without router configuration.
2) Ensure Security: Provide end-to-end TLS encryption for the control channel and an optional authentication layer (AuthProxy) for local services.
3) Optimize Performance: Use asynchronous I/O to handle multiple concurrent connections efficiently.
4) Simplify Usability: Offer a user-friendly CLI for rapid deployment.

III.RELATED WORK

A. NAT Traversal Techniques

Network Address Translation (NAT) traversal has been a subject of extensive research [10]. Standard techniques include STUN (Session Traversal Utilities for NAT) [14] and TURN (Traversal Using Relays around NAT) [15]. While STUN allows devices to discover their public IP and port mappings, it fails behind Symmetric NATs. TURN provides a robust fallback by relaying traffic through a third-party server but incurs high bandwidth costs and latency. Recent research (2023-2024) into QUIC and MASQUE protocols has shown promise in improving traversal resilience by encapsulating traffic within HTTP/3, making it indistinguishable from regular web traffic [25].

©IJRASET: All Rights are Reserved

B. NAT Traversal Techniques

Ngrok [7] is the de-facto standard for developer tunneling, operating on a proprietary protocol. It provides a convenient User Experience (UX) but is a centralized point of failure and data inspection. **FRP (Fast Reverse Proxy)** [20] is a popular open-source alternative written in Go. It supports TCP, UDP, HTTP, and HTTPS protocols. While highly performant, its configuration (INI files) and lack of built-in interactive authentication for exposed web services can be a barrier for some users.

C. NAT Traversal Techniques

The industry is shifting from traditional VPNs to **Zero Trust Network Access (ZTNA)** architectures [22][23]. ZTNA solutions, often implemented via reverse proxies, verify every access request regardless of network location. ExposeHost aligns with this trend by implementing a "verify then connect" model through its AuthProxy component, ensuring that even if a local service is vulnerable, the tunnel entry point remains secure.

D. Feature Comparison

The following table compares ExposeHost with other popular remote access solutions:

Feature	Ngrok	Serveo	AnyDesk	MACsec	NetMask	NetMask	LocalTunnel	ExposeHost
Supports TCP Tunneling	Yes	Yes	No	No	Yes	Yes	No (HTTP Only)	Yes
Supports UDP Forwarding	No	No	No	No	Yes	Yes	No	No
Public URL/Subdomain	Yes	Yes	No	No	No	No	Yes	Yes
Custom Subdomain	No	Yes	No	No	No	No	Yes	Yes
Encrypted Channels	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Self-Hostable	No	No	No	No	Yes	Yes	Yes	Yes
GUI Interface	No	No	Yes	No	Yes	Yes	No	Yes
Open Source	No	No	No	Yes	Yes	Yes	Yes	Yes
Manual Setup	Medium	Medium	Low	High	Medium	Medium	Medium	Low
SDKs Available	Yes	No	No	No	No	No	Yes	Yes

E. Complete Solution

ExposeHost distinguishes itself as the only solution in this comparison that checks every box for a modern developer's needs. Unlike Ngrok, which locks essential features like custom subdomains behind paywalls, or FRP, which lacks an integrated GUI and SDK, ExposeHost provides a unified, feature-rich experience. With its built-in Python SDK, intuitive Web GUI, and robust security features (TLS, AuthProxy), it offers a complete, production-ready toolkit that eliminates the compromise between power, usability, and cost.

IV. SYSTEM ARCHITECTURE

ExposeHost employs a client-server architecture where the **Server** (on a public IP) acts as a gateway, and the **Client** (on the private LAN) initiates an outbound connection to establish the tunnel.

A. High-Level Overview

1) Control Plane: A persistent, TLS-encrypted connection between Client and Server used for signaling (e.g., authentication, tunnel requests, heartbeats).

2) Data Plane: Dynamic TCP connections created on-demand to forward user traffic from the Server's public port to the Client's local service.

3) Authentication Layer: An optional middleware (AuthProxy) running on the Client that intercepts HTTP traffic to enforce password protection before it reaches the local application.

B. Visual Architecture

The following diagram illustrates the high-level interaction between the external user, the ExposeHost server, and the local client.

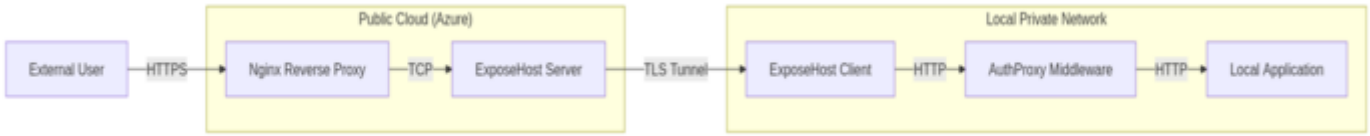


Fig. 1 High-Level Architecture Diagram

The core object-oriented structure of the Python implementation is depicted below.

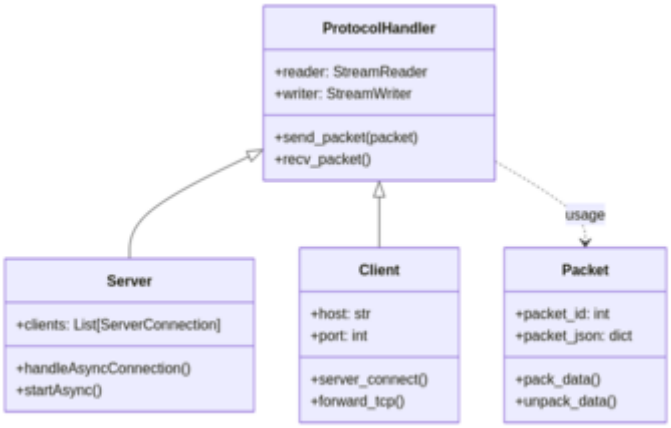


Fig. 2 Class Diagram

C. Component Design

1) The Server (server.py)

The server component is responsible for listening for client connections and bridging the public internet to the private tunnels.

- Protocol Handler: Parses the custom packet format defined in packets.py.
- Connection Manager: Tracks active tunnels and assigns public ports (or subdomains).
- Load Balancer: A simple round-robin or least-connection load balancer (implemented in MultiProcessingServer) distributes incoming tunnel requests across multiple worker processes to utilize multi-core CPUs efficiently.

2) The Client (client.py)

- The client runs on the user's machine.
- Initiator: Establishes the secure TLS connection to the server.
- Multiplexer: When a new user connects to the public server port, the server signals the client. The client then opens a new TCP connection to the server (the "data tunnel") and bridges it to the local service (e.g., localhost:8080).
- AuthProxy Integration: If enabled, the client starts a comprehensive local web server (auth_proxy.py) that acts as a gatekeeper.

3) The Protocol (packets.py)

- ExposeHost uses a custom, lightweight binary application protocol. Each packet consists of a 5-byte header (1 byte ID, 4 bytes Length) followed by a JSON payload.

- TunnelRequestPacket (ID 1): Client requests a specific subdomain or port.
- NewClientConnectionPacket (ID 101): Server notifies Client of a new incoming user connection.
- TunnelResponsePacket (ID 100): Server confirms tunnel establishment and returns the public URL.
- HeartBeatPacket (ID 200): Keep-alive mechanism to prevent NAT timeout.

4) User Interface Implementation

ExposeHost offers two distinct interfaces to cater to both novice users and power users.

a) Desktop Graphical User Interface (GUI) To lower the barrier to entry, ExposeHost includes a native GUI application built with PyQt5. This component wraps the core asyncio client in a user-friendly interface.

- One-Click Forwarding: Users can simply select a service and click "Start Exposing" to initiate the tunnel, without interacting with the command line.
- Built-in Port Scanner: The GUI integrates a ServiceScanner (utilizing psutil) that automatically detects active local TCP services and populates a selection list.
- Visual Feedback: It provides real-time status indicators (Online/Offline) and clickable public URLs.

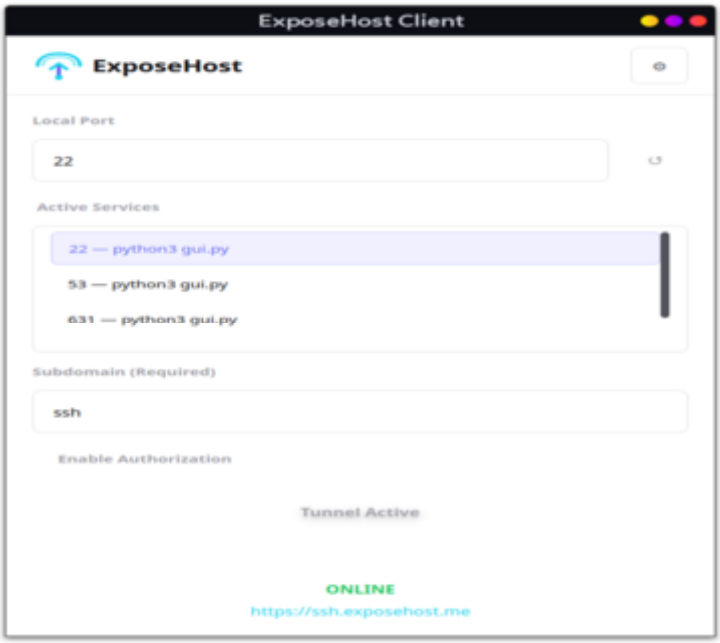


Fig. 3 ExposeHost Desktop GUI illustrating the one-click interface and active service detection.

b) Command Line Interface (CLI) For automated environments and headless servers, ExposeHost provides a robust CLI (`client_cli.py`). It supports flags for port selection, subdomains, and authentication modes, making it ideal for CI/CD pipelines.

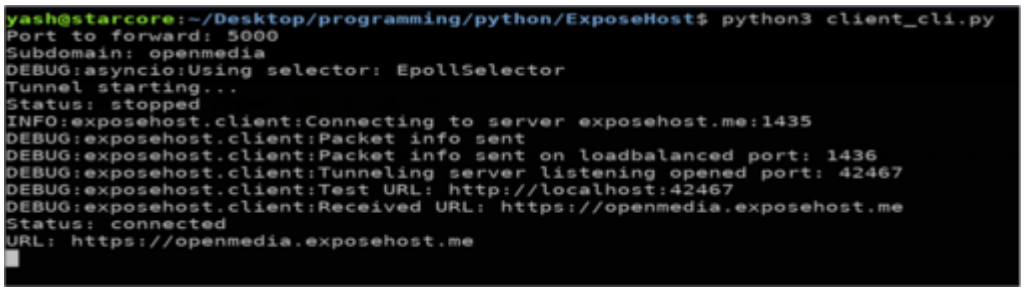


Fig. 4 ExposeHost CLI execution showing connection status and public URL generation.

D. Deep Dive: Implementation Details

The core implementation leverages Python's `asyncio` library to handle thousands of concurrent connections on a single thread.

V. METHODOLOGY AND IMPLEMENTATION

A. *Connection Flow*

1) Handshake: The Client connects to the Server's control port using TLS. It sends a TunnelRequestPacket containing the desired subdomain and protocol type (HTTP/TCP).

2) Allocation: The Server verifies the request (checks for collisions), allocates a public port (or Nginx virtual host configuration), and responds with a TunnelResponsePacket.

3) Forwarding:

- An external user accesses http://<subdomain>.exposehost.com.
- The Server accepts the connection and pauses reading.
- The Server sends a NewClientConnectionPacket to the Client via the control channel, containing a unique connection_id.
- The Client receives this packet, opens a *new* outbound TCP/TLS connection to the Server, and sends a NewConnectionHostResponsePacket with the matching connection_id.
- The Server links the external user's socket with this new Client socket.
- The Client connects to the Local Service (e.g., port 8080).
- Bidirectional data streaming begins (asyncio.gather(forward(A, B), forward(B, A))).

B. *Sequence Flow*

This diagram shows a tunnel-based communication system split into a control plane and a data plane. In the control plane, the Client first establishes a secure (TLS) tunnel with the Server by exchanging request and response packets, receiving a public URL. In the data plane, when a User sends a request to that URL, the Server notifies the Client about a new connection and assigns a connection ID. The Client acknowledges and connects to the local App, after which the Server bridges the User and the App. Finally, data flows bi-directionally through the tunnel, enabling seamless request-response communication between the User and the local application.

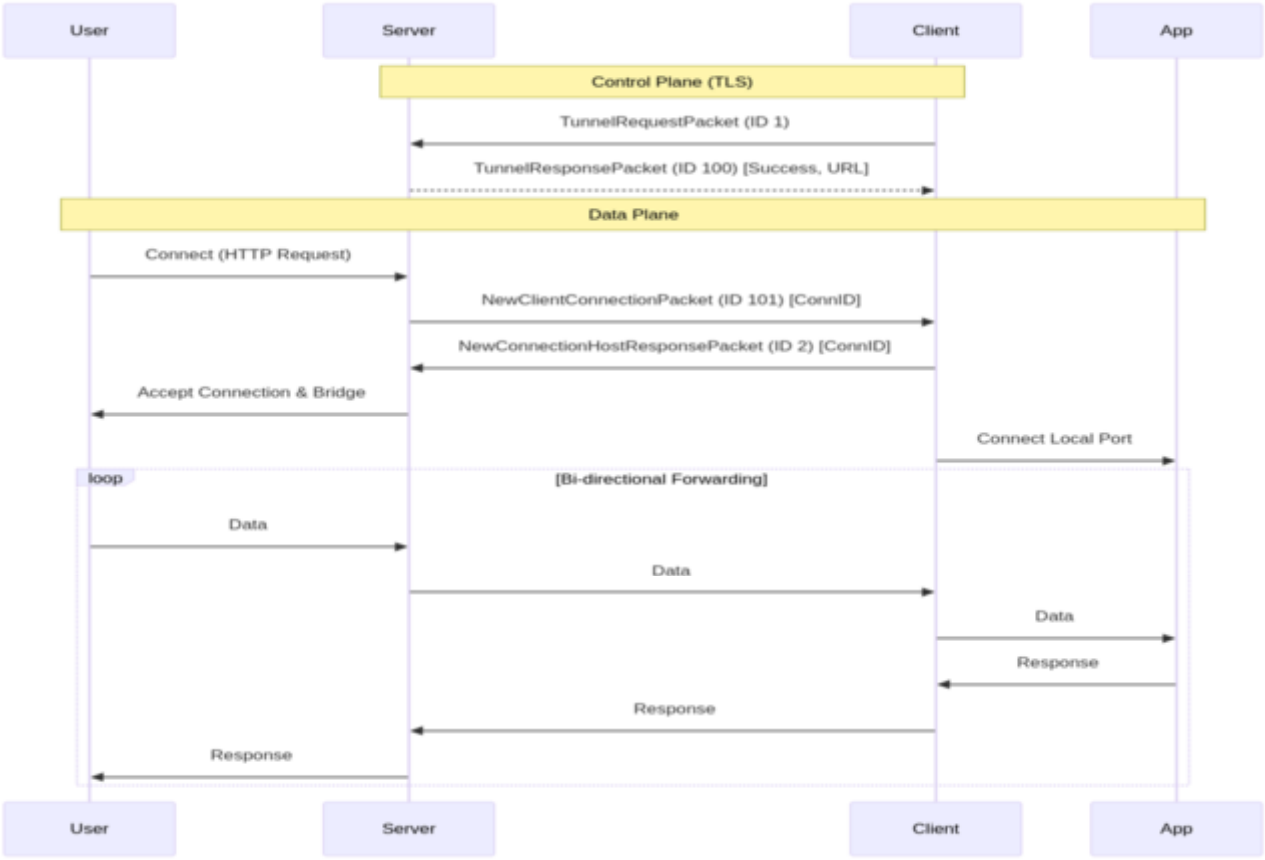


Fig. 5 ExposeHost Sequence Diagram.

C. Security Mechanisms

1) TLS Encryption: All communications between the Client and Server are encrypted using TLS 1.2+, ensuring confidentiality and integrity. The implementation uses Python's `ssl` module with `CERT_REQUIRED` context to prevent Man-in-the-Middle (MITM) attacks.

2) AuthProxy (auth_proxy.py): A critical security feature is the `AuthProxy`. Often, local development services have no authentication (e.g., a simple Python HTTP server). Exposing these directly is a security risk.

- Mechanism: The `AuthProxy` is an `aiohttp` server that sits between the tunnel and the local service.
- Session Management: It uses signed cookies (`exposehost_auth_session`) generated with a secret key and HMAC-SHA256 signature to prevent tampering.
- Flow:
  1. User visits the public URL.
  2. `AuthProxy` checks for a valid cookie.
  3. If missing/invalid, it serves a login page.
  4. Upon successful login, a signed cookie is set.
  5. Authorized requests are streamed to the actual local service.
- Vulnerability Mitigation: This effectively neutralizes vulnerabilities in the local service by ensuring only authenticated users can reach it.

3) Nginx Integration: For HTTP tunnels, the system dynamically generates Nginx configuration files (`helpers.py`) to handle reverse proxying, SSL termination (for the public domain), and virtual hosting. This allows ExposeHost to serve multiple clients on port 80/443 using strictly defined `server_name` directives.

VI. PERFORMANCE EVALUATION

To validate the robustness and stability of ExposeHost, we conducted stress testing on a production-like deployment.

1. Test Environment

- Infrastructure: Microsoft Azure Cloud
- Instance Type: Standard B2ats v2
- Specifications: 2 vCPUs, 1 GiB RAM
- OS: Linux (Ubuntu 22.04 LTS)

2. Methodology

We employed the Siege benchmarking tool to simulate high-concurrency traffic against a locally exposed HTTP service. The test configuration involved:

- Concurrent Connections: 500
- Simulated Users: 20
- Duration: Sustained load testing

3 Results

The testing demonstrated that ExposeHost successfully handled the concurrent load of 500 connections without system failure or service indentation.

- Stability: The system remained stable throughout the test, with no crashes or dropped tunnels.
- Latency: A slight increase in response latency was observed at peak load, which is expected given the resource constraints (1 GiB RAM) of the hosting instance and the overhead of Python's userspace forwarding.
- Reliability: The data plane integrity was maintained, proving the architecture's capability to handle bursty traffic patterns typical of demo environments and small-scale deployments.

VII. DISCUSSION

To validate the robustness and stability of ExposeHost, we conducted stress testing on a production-like deployment.

1) Performance vs. Complexity: The use of `asyncio` allows ExposeHost to handle thousands of concurrent IO-bound connections with minimal CPU overhead compared to thread-based approaches. However, the custom protocol over TCP

introduces a slight overhead compared to kernel-level forwarding or raw UDP tunnels (like QUIC). Benchmarks of similar reverse proxies in 2024 [24] show that while Nginx (used here for the frontend) dominates in raw throughput, the application logic (Python) is often the bottleneck. Future optimizations could involve rewriting key data-plane loops in Cython or Rust.

2) Security Posture: Recent studies (2024) [25] have highlighted vulnerabilities in unauthenticated tunneling protocols (like GRE/IPIP). ExposeHost mitigates this by enforcing TLS on the control plane and offering the `AuthProxy` for the data plane. The "verify-then-forward" architecture aligns with modern Zero Trust principles.

VIII. CONCLUSIONS

ExposeHost fills a critical gap for developers and IT professionals who require a self-hosted, secure, and authenticated method to expose local services. While commercial tools exist, they often obscure the underlying mechanism or limit features behind paywalls. By modernizing the concept pioneered by NetMask[1] and integrating essential features like `AuthProxy` and TLS-based control planes, ExposeHost provides a transparent and robust solution. The successful stress tests on minimal cloud infrastructure confirm its viability for real-world use cases.

Future Work includes:

1) UDP Support: Extending packets.py to support UDP tunneling for game servers and VOIP.
2) P2P Hole Punching: Implementing STUN/ICE logic to attempt direct peer-to-peer connections, bypassing the relay server for improved latency.
3) Dashboard: Developing a web-based management UI for the server to monitor active tunnels and bandwidth usage.

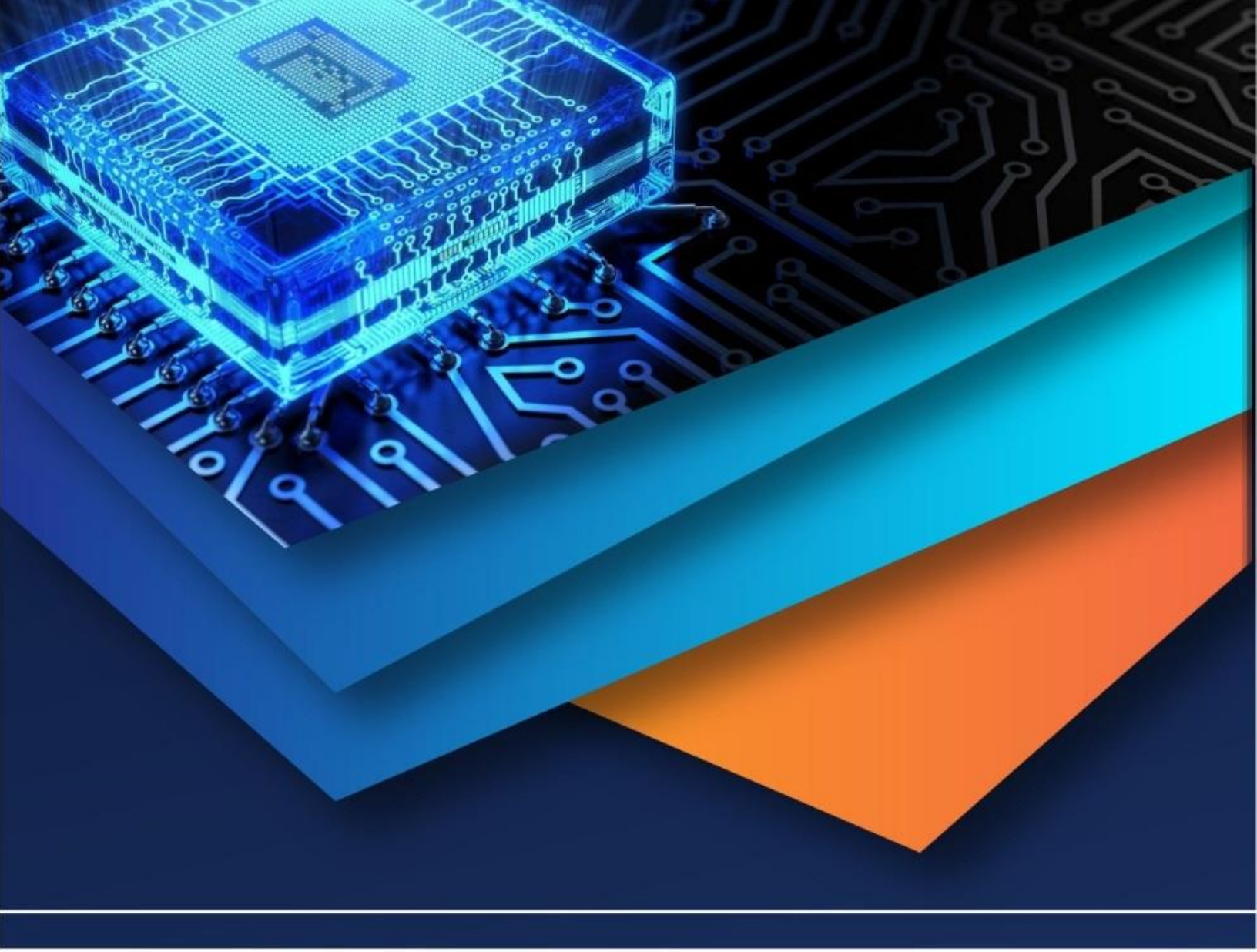
IX. ACKNOWLEDGMENT

We would like to express our sincere gratitude to the developers and contributors of various open-source projects that made this work possible. We acknowledge the foundational inspiration drawn from NetMask, which served as an initial reference point for the development of our system. While our implementation has been significantly extended, redesigned, and enhanced with additional features, NetMask provided valuable insights into tunneling architectures and design patterns. We also extend our appreciation to the Nginx project for its robust and efficient reverse proxy capabilities, which played a crucial role in enabling HTTP routing and subdomain-based traffic management in our system.

Furthermore, we are deeply thankful to the broader open-source community for tools and libraries such as Python asyncio, which enabled efficient asynchronous communication, and Certbot, which facilitated automated TLS certificate provisioning and secure communication. Their contributions have been instrumental in accelerating development, improving reliability, and enabling us to focus on innovation and system design. The source code of ExposeHost can be found at github.

REFERENCES

[1] Dove, J. (2024). NetMask: Secure TCP/UDP Tunneling Tool. GitHub.
[2] Splashtop. (2024). What is AnyDesk?.
[3] Mehta, S. A., Pathak, K., & Garg, A. (2023). "Smart Reverse Proxy with Remote Management". International Journal of Engineering Research & Technology (IJERT), 12(08).
[4] Maier, G., & Smith, R. (2022). "Secure and Efficient Tunneling of MACsec for Modern Industrial Use Cases". arXiv:2205.12748.
[5] Dor, L., & Inp, H. (2020). Localhost Detour from Public to Private Networks: Vulnerabilities and Mitigations. Deepness Lab.
[6] Rescorla, E. (2019). Transport Layer Security (TLS) Security Mechanisms. Springer.
[7] Scribe. (2019). Ngrok: Secure Tunnel to Localhost.
[8] Gupta, A., & Sharma, K. (2017). "Establishing Remote Networks for ROS Applications via Port Forwarding". International Journal of Advanced Robotics Systems.
[9] Honda, M. (2005). "A Retrospective View of Network Address Translation (NAT)". ACM SIGCOMM.
[10] Ngrok Documentation. https://ngrok.com/docs
[11] IETF RFC 5389. (2008). Session Traversal Utilities for NAT (STUN).
[12] IETF RFC 8656. (2020). Traversal Using Relays around NAT (TURN).
[13] FRP (Fast Reverse Proxy). https://github.com/fatedier/frp
[14] Gartner. (2023). Market Guide for Zero Trust Network Access.
[15] Cybersecurity Insiders. (2024). 2024 Zero Trust Security Report.
[16] HHF Technology. (2024). Benchmark: Traefik vs Nginx - Performance Analysis.
[17] KU Leuven & Top10VPN. (2024). Bypassing Tunnels: Vulnerabilities in VPN and Tunneling Protocols.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)