NETCLOAK (EMPHASIZED DATA CONCEALMENT WITHIN NETWORK PROTOCOLS)

ABSTRACT

The widespread use of computer networks has led to the development of advanced techniques for hiding information within transmitted data, a practice known as network steganography. Among various protocols, HTTP stands out as an effective medium for embedding covert data due to its role in handling web application requests and responses. This work explores network steganography techniques that leverage HTTP-specific properties, focusing on concealing data within HTTP headers. We present an evaluation framework to assess the efficiency, security, and detectability of these methods, supported by experimental results that highlight the practical applications and effectiveness of hidden communication within web traffic.

Keywords: covert communication, detectability, HTTP protocol, network steganography, web traffic, etc.

Table of Contents

ACKNOWLEDGEMENT ..... **Error! Bookmark not defined.**

ABSTRACT II

Table of Contents V

LIST OF FIGURES VII

LIST OF TABLES VIII

LIST OF ABBREVIATIONS IX

1. INTRODUCTION 1

1.1 Introduction 1

1.2 Objective of the study 1

1.3 Describe Steganography 1

2. BACKGROUND AND LITERATURE REVIEW 2

2.1 Introduction 2

2.2 Previous Research 2

2.3 Classification of Network Steganography 3

2.4 Steganalysis Techniques 4

3. REQUIREMENTS AND WORKING PRINCIPLES 6

3.1 Introduction 6

3.2 Steganographic Methods 6

3.3 Tools and Requirements 13

3.4 Data Concealment 14

3.5 Considerations for Implementing 15

3.6 Security and Ethical Considerations 17

3.7 Proposed Model 18

3.8 Explanation of Tools & Code Structure 20

3.9 Compression and Encryption Techniques 21

4. PROPOSED METHOD 22
4.1 Backend Configuration 22
4.2 Frontend Configuration 24
4.3 Flowchart 25
4.4 Real-World Applications and Case Studies 29
4.5 Previous Research 30
5. EXPERIMENTAL RESULTS AND ANALYSIS 31
5.1 Web Interface 31
5.2 Comparative Analysis 33
5.3 Expanded Comparative Tables and Graphs 36
5.4 Performance Metrics: Memory, CPU Usage 37
6. RESULTS & DISCUSSIONS 39
6.1 Testing Strategy: Unit, Integration & System Testing 39
6.2 Performance Evaluation 40
6.3 Comparisons with Existing Methods 40
7. CONCLUSION & FUTURE SCOPE 42
7.1 Conclusion 42
7.2 Limitations 42
7.3 Future Enhancement 43
BIBLIOGRAPHY 44
APPENDICES 45
PUBLICATIONS ..... **Error! Bookmark not defined.**
PLAGARISM REPORT ..... **Error! Bookmark not defined.**

LIST OF FIGURES

Figure No.	Figure Caption	Page No.
3.1	Process followed in HICCUPS	7
3.2	Influence on frame error rate in HICCUPS	7
3.3	Transmission flow in RSteg	7
3.4	Step 1 in TranSteg	10
3.5	Step 2 in TranSteg	10
3.6	Step 3 in TranSteg	10
3.7	StegSuggest method	12
3.8	Process involved in StegSuggest method	12
3.9	Proposed Model	19
4.1	Flowchart	28
5.1	Sender Interface	31
5.2	Receiver Interface	32
5.3	Dashboard Interface	33
5.4	Data Input Capacity vs Success Rate	34
5.5	Normal vs Encrypted Packet Size	34
5.6	Storage Capacity vs Failure Threshold	35

LIST OF TABLES

Table No.	Table Caption	Page No.
3.1	Encryption Alternatives	21
5.1	Enhanced Protocol Analysis Table	35
5.2	Evaluating Network Steganography Methods	36
5.3	System Performance Analysis	38

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
API	Application Programming Interface
APT	Advanced Persistent Threat
CIA	Central Intelligence Agency
CNN	Convolutional Neural Network
DDoS	Distributed Denial of Service
DPI	Deep Packet Inspection
DNS	Domain Name System
DNScat2	DNS-Based Tunneling Tool
DoS	Denial of Service
EU	European Union
FER	Frame Erasure Rate
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IoT	Internet of Things
IP	Internet Protocol
LAN	Local Area Network
LACK	Lost Audio Packets Steganography
PDU	Protocol Data Unit
QUIC	Quick UDP Internet Connections
SCTP	Stream Control Transmission Protocol
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTL	Time To Live
UDP	User Datagram Protocol
VPN	Virtual Private Network

Chapter 1

INTRODUCTION

1.1 Introduction

In the digital era, protecting information has become increasingly complex due to growing security threats. Frameworks like the CIA Triad and Parkerian Hexad help define key aspects of data security, ensuring confidentiality, integrity, and availability. This study explores network steganography, a technique that conceals data within network protocols to enable discreet communication. Unlike cryptography, which makes data unreadable through encryption, steganography hides information within ordinary traffic, making it appear harmless. The difference between information hiding and message hiding is often misunderstood, leading to confusion in security discussions. Network steganography techniques vary depending on how they embed data and the type of carrier medium used, providing multiple approaches to secure covert communication.

1.2 Objective of the study

- Explore, develop, and evaluate techniques for embedding hidden data within standard network protocols without detection.
- Investigate the use of DNS, TCP, and HTTP manipulation as effective steganographic approaches, given their widespread use in digital communication.
- Examine practical applications, such as securing sensitive exchanges, enabling confidential communication, and ethically deploying covert data transmission methods.

1.3 Describe Steganography

Derived from the Greek words “Steganos” (meaning “covered”) and “Graphos” (meaning “writing”), steganography literally translates to covered writing. This ancient practice has long been employed for discreet communication, allowing two parties to exchange secret messages without drawing suspicion.

The key principle of steganography is ensuring that an unintended observer remains unaware that communication is occurring. If a third party detects the presence of hidden data, the purpose of steganography is compromised, and the method fails. Successful execution requires a natural-looking and unsuspecting carrier medium to embed concealed information without raising suspicion.

Chapter 2

BACKGROUND AND LITERATURE REVIEW

2.1 Introduction

Jun O Seo et al. [1] proposed a structured approach for designing network steganography, built around three key processes: location identification, steganogram concealment, and validation.

- Location Identification focuses on understanding network protocols and pinpointing areas that can be exploited as covert communication channels.
- Steganogram Concealment involves embedding secret data within network packets, categorized into storage-based and timing-based techniques.
- Validation ensures that the steganographic method functions as intended while adhering to core principles of network steganography.

This framework provides a systematic way to execute covert communication while minimizing the risk of detection.

2.2 Previous Research

Several researchers have proposed different methods for implementing network steganography, each with unique approaches and challenges.

- **StegBlock Approach** (Fraczek & Szczypiorski [2]): This method enables covert communication by embedding hidden identifiers within StegBlocks—sequentially organized units where secret message bits are encoded through the final bits of each object count. Inspired by Cachin's principles of undetectability, this approach minimizes anomaly detection risks without increasing network traffic. It utilizes Packet Payload Byte Parity and Alternative Packet Payload Bit Parity as covert channels, ensuring resilience against detection mechanisms, albeit with low bandwidth.

- **UDP Datagram Modification** (Nair et al. [3]): A more secure technique that conceals secret data by modifying UDP datagram lengths. Unlike previous length-based embedding strategies, this method leverages the random pattern behavior of UDP datagram lengths, making it highly suitable for steganographic purposes. Results suggest that the embedded messages seamlessly blend into normal traffic flow, reducing detection risk.

- **Packet Loss Concealment for VoIP** (Aoki [4]): An innovative approach that embeds forward pitch variations within packets as part of VoIP communications. This technique helps recover dropped packets by approximating missing data based on the preceding packet, ensuring smoother transmission while enabling hidden communication.

- **Transcoding-Based Steganography** (Mazurczyk et al. [5]): This method increases steganographic storage capacity by replacing an original codec with a visually similar alternative. By maintaining the appearance of the original codec while embedding secret data, this technique enhances detectability resistance in environments with limited bandwidth.

- **SCTP-Based Steganography** (Fraczek [6]): This approach exploits key features of Stream Control Transmission Protocol (SCTP)—namely multihoming and multistreaming—for hidden data transmission. In multihoming, packets sent from one IP address are treated as '0', while packets from another IP are treated as '1'. In multistreaming, each stream is assigned a bit for encoding purposes, and streams are modified post-embedding.

Additionally, various methods embed secret data into the identification field of IP packet headers, allowing up to 16 bits per packet. However, these approaches face challenges such as data loss due to fragmentation or repetition errors caused by identical IP IDs in fragmented packets—leading to potential inaccuracies in steganographic transmission.

2.3 Classification of Network Steganography

Network steganography can be categorized based on protocol layers, modification strategies, and covert channel behaviours:

a) Classification by Protocol Layer:

- **Application Layer (e.g., HTTP, DNS, VoIP):** Alters headers, payloads, or query structures to conceal data.
- **Transport Layer (e.g., TCP/UDP):** Embeds secret data within TCP flags, sequence numbers, or datagram lengths.

- **Network Layer (e.g., IP):** Uses IP header fields (such as IP ID and TTL) to encode hidden messages.
- **Link Layer (e.g., Ethernet):** Exploits Ethernet padding bits for steganographic storage.

b) Classification by Modification Type:

- **Payload-Based Steganography:** Embeds data within packet payloads.
- **Header-Based Steganography:** Alters protocol header fields to store hidden information.
- **Timing-Based Steganography:** Encodes data through variations in packet transmission timings.

c) Classification by Covert Channel Behaviour:

- **Storage Channels:** Embeds hidden data into existing protocol fields.
- **Timing Channels:** Manipulates packet timing patterns to transmit secret messages.
  - **Example:** HICCUPS modifies corrupted frames (storage), while LACK alters audio delays (timing).

2.4 Steganalysis Techniques

Steganography detection requires steganalysis, which employs various techniques to identify hidden communication patterns. Some of the primary detection methods include:

a) Signature-Based Detection:

- Recognizes predefined patterns or steganographic signatures in packets.
- **Pros:** Quick detection for known methods.
- **Cons:** Limited effectiveness against novel techniques.

b) Statistical Anomaly Detection:

- Analyzes network traffic for irregularities in:
  - **Packet sizes**
  - **Transmission timing**
  - **Entropy in protocol headers**
- Helps detect anomalies such as unusual TCP flag sequences or oversized DNS queries.

c) Machine Learning-Based Detection:

- Uses AI models to differentiate normal network traffic from steganographic transmission.
- **Key Features:** Packet frequency, header field modifications, timing inconsistencies.
- **Common ML Tools:** Support Vector Machines (SVM), Random Forest, and Convolutional Neural Networks (CNN).
- **Example:** Machine learning-based Intrusion Detection Systems (IDS) flag irregular HTTP headers.

d) Visual & Heuristic Approaches:

- Applied primarily to application-layer steganography techniques, such as covert data embedding in web search suggestions (Google Suggest) or VoIP packets.
- Focuses on identifying unusual **GUI behaviour or packet sequences** indicative of hidden communication.

Chapter 3

REQUIREMENTS AND WORKING PRINCIPLES

3.1 Introduction

Steganography is a technique used to embed hidden information within publicly accessible data, such as digital media, for purposes like copyright protection and secure communication. Network steganography, in particular, enables the creation of covert communication channels within existing open networks, allowing metadata to be transmitted discreetly without generating additional network traffic.

Steganographic data transmission can occur across multiple layers of network communication, making the hidden data flow dependent on regular traffic generated by other programs and applications. Methods of network steganography typically leverage optional fields in standard protocols, redundant data, or specific interpretations of protocol data units (PDUs).

Examples of these techniques include embedding data within TCP segments, TCP reserved fields, and HTTP protocols. While these methods enable discreet data distribution, they require continuous monitoring of established connections to ensure effectiveness and security.

3.2 Steganographic Methods

3.2.1 HICCUPS (Hidden Communication System for Corrupted Networks)

The HICCUPS method operates by sending intentionally corrupted frames, prompting network stations to adjust their mode based on the received corrupted frame. In this context, a "corrupted frame" refers to a frame containing an incorrect checksum.

HICCUPS falls under protocol data unit (PDU) modifications, where payload alterations are made according to the system’s requirements. As illustrated in Figure 3.1, primary data is transmitted alongside intentionally corrupted frames, effectively embedding a hidden message within the communication stream. Upon reception, the hidden message is reconstructed, allowing secret data retrieval.

However, the introduction of corrupted frames inevitably leads to frame errors, quantified by the Frame Error Rate (FER). FER is calculated as the ratio of erroneous data packets to the total number of received packets. If this ratio becomes too high, the connection might drop,

disrupting communication. For HICCUPS to function effectively, the FER must remain within an optimal range, ensuring continuous data transmission.

As shown in Figure 3.2, a comparative analysis between a standard communication network and a HICCUPS-based network illustrates the FER variations, highlighting how intentional corruption impacts network behavior while maintaining hidden data transmission.

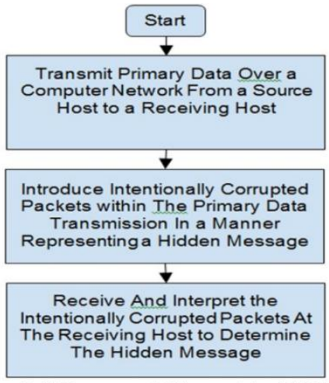

Figure 3.1 Process followed in HICCUPS

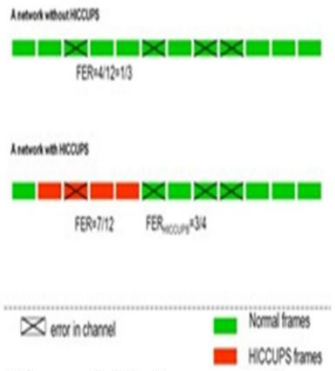

Figure 3.2 Influence on frame error rate in HICCUPS

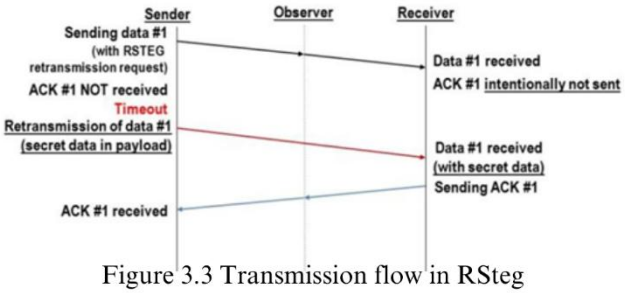

Figure 3.3 Transmission flow in RSteg

3.2.2 LACK (Lost Audio Packets Steganography)

This method also falls under protocol data unit (PDU) modifications. It works by generating voice packets at the transmitter end, with one packet deliberately delayed. The payload of this selected packet is replaced with a delayed steganogram, embedding hidden data within the transmission.

Once the delay timer expires, the modified packet is sent to the receiver. For a standard receiver, this delayed packet appears lost and is discarded. However, a LACK receiver is designed to recognize and extract the hidden steganogram from the delayed voice packet, allowing successful covert communication.

3.2.3 RSTEG (Retransmission Steganography)

In this method, a retransmission mechanism is used to embed hidden data. The receiver intentionally acknowledges that certain received packets were not received, prompting the transmitter to resend them. During retransmission, the payload of the packet is replaced with a hidden steganogram, allowing covert data exchange.

This technique is highly difficult to detect because it mimics legitimate network behavior. As illustrated in Figure 3.3, the sender initially transmits data #1, but the receiver falsely indicates non-receipt, preventing the sender from receiving an acknowledgment. As a result, the sender retransmits the packet, embedding secret data within the payload. Once the receiver successfully receives the retransmitted packet, an acknowledgment is sent, completing the process.

3.2.4 SCTP (Stream Control Transmission Protocol)

This method is based on multi-streaming, where the steganogram is split into smaller parts rather than being transmitted as a single unit. Each segment is then sent across different streams, ensuring discreet and efficient data transmission.

For example, if the complete steganogram is 10011100, it is distributed across four streams (1, 2, 3, and 4):

- Stream 3 carries 10, corresponding to that section.
- Stream 2 carries 01.
- Stream 4 carries 11.
- Stream 1 carries 00.

By fragmenting the data this way, the hidden message is seamlessly embedded into multiple communication channels, making detection significantly more challenging. The robustness of this method increases when UDP and TCP components are combined, as they provide additional flexibility and variability in data transmission.

3.2.5 PadSteg (Padding Steganography)

PadSteg is a type of inter-protocol steganography, meaning it utilizes multiple protocols within the TCP/IP stack for covert communication. In this method, the padding bits of Ethernet frames

are replaced with the hidden steganogram, embedding secret data within standard network transmissions.

Due to a phenomenon known as Etherleak, a regular receiver typically does not prioritize detecting PadSteg, as the presence of padding bits is considered normal. Etherleak occurs due to variations in padding standardization, meaning different systems handle padding inconsistencies differently, inadvertently allowing PadSteg to remain unnoticed during regular network communication.

3.2.6 TranSteg (Transcoding Steganography)

Similar to LACK, this method also leverages IP telephony for covert communication. TransSteg is considered one of the most efficient VoIP-based steganographic methods because, unlike other techniques, it not only allows the extraction of the steganogram at the receiver end but also enables the restoration of the original data—a key advantage.

The process begins with compressing open data, creating space within the packet for the hidden steganogram. At the receiver's end, the steganogram is extracted, and the original voice data is restored without loss. As shown in Figure 3.4, the typical voice data content in an original VoIP packet ranges between 20-30ms, which is then compressed during transcoding to accommodate the secret message.

One challenge with compression is that it modifies the checksum, potentially altering packet integrity. To counter this, the secret message is embedded without changing the original checksum, ensuring seamless transmission. After embedding the hidden data, the transcoded audio packet is sent to the receiver, assuming Alice as the sender and Bob as the recipient. At the sender’s end, the RTP payload size is intentionally kept larger than the inserted voice payload, as illustrated in Figure 3.5.

For message extraction, the receiver identifies and retrieves the steganogram from the transcoded packet, reconstructing the original data. Figure 3.6 visually demonstrates this extraction process, showcasing how the hidden message is recovered without compromising the transmitted data.

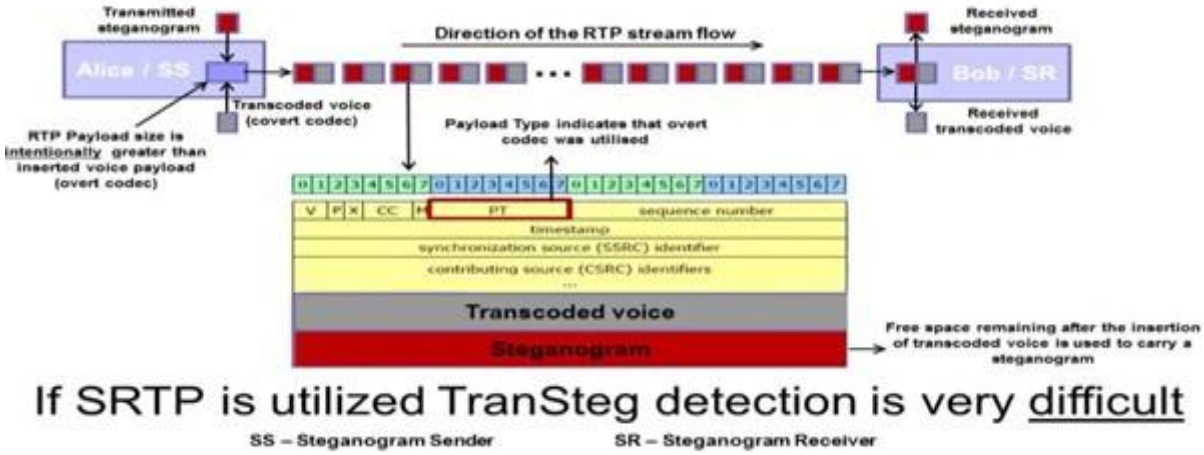

Figure 3.4: Step 1 in TranSteg

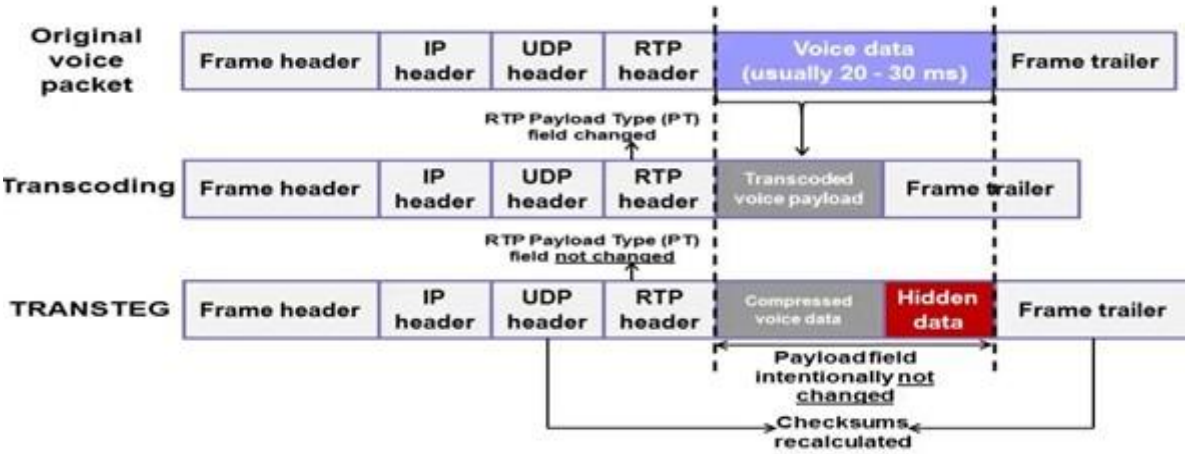

Figure 3.5: Step 2 in TranSteg

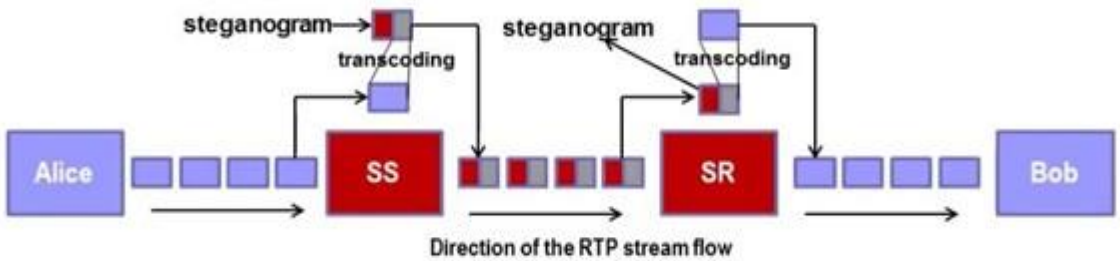

Figure 3.6: Step 3 in TranSteg

3.2.7 SkyDe (Skype Hide)

SkyDe is a steganographic technique that leverages Skype's peer-to-peer (P2P) IP telephony for covert data transmission. During Skype communication, several silent packets—packets that contain no voice signals—are generated naturally within the stream. SkyDe replaces these silent packets with hidden messages, embedding secret data within the existing communication flow.

This method introduces minimal distortion in the transmission, ensuring the integrity of the original data while discreetly inserting steganographic content. Additionally, SkyDe offers high bandwidth for data transmission, making it an efficient and reliable technique for concealed communication. Due to its seamless integration into regular Skype traffic, detecting SkyDe is extremely difficult, further enhancing its effectiveness in secure data exchange.

3.2.8 StegTorrent

StegTorrent is a steganographic method built on a peer-to-peer (P2P) data exchange protocol, allowing a single client to share files with multiple recipients simultaneously over IP networks. Unlike standard P2P file-sharing, StegTorrent enables covert communication between a sender and a receiver who are already familiar with each other's IP addresses.

To transmit secret information, the sender uses a modified BitTorrent client, embedding hidden data within shared files. The receiver accesses the hidden message using a dedicated StegTorrent client, ensuring the secure retrieval of covert data without disrupting normal file-sharing operations.

3.2.9 StegSuggest

This method utilizes Google Suggestions as a carrier for hidden messages. Whenever a user types a search term, Google automatically generates suggestions based on predictive algorithms. The secret message is concealed by suffixing a letter to each word in the suggestion list, effectively embedding hidden data within the search suggestions.
Since Google Suggest operates on AJAX, data retrieval occurs asynchronously. Each time a user enters a search query, the Google client sends an HTTP GET request to the Google server, fetching relevant suggestions. This mechanism allows steganographic data transmission by modifying the suggestions with a subtle encoding strategy.

As illustrated in Figure 3.7, each letter typed results in a corresponding HTTP GET request, generating a new suggestion list from the server. During this exchange, a steganogram sender (SS) and receiver (SR) participate in covert communication, where the Google Suggest server serves as the carrier for hidden information. To ensure successful transmission, both SS and SR must be on the same network node.

Every time SR searches for a term, SS embeds a word within each suggestion, gradually transmitting the complete secret message. As shown in Figure 3.8, this process allows discreet data exchange through search suggestion manipulation.

Figure 3.7: StegSuggest method

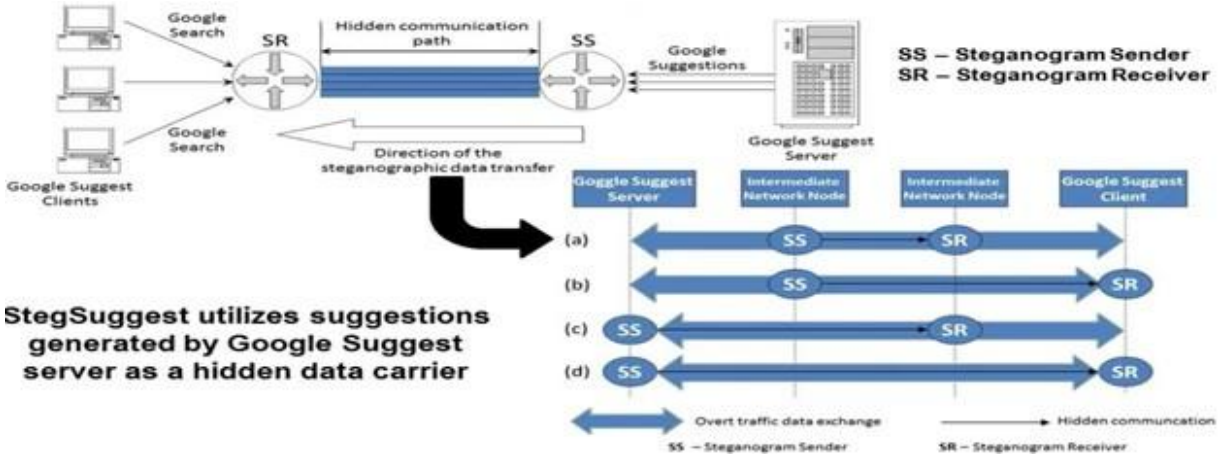

Figure 3.8: Process involved in StegSuggest method

3.3 Tools and Requirements

3.3.1 External Tool: Npcap

Npcap is a powerful packet capture and network analysis tool designed for Windows operating systems. It consists of a software library and a network driver, enabling users to capture and inspect network packets in real time. This level of deep traffic analysis is particularly useful in network steganography, where hidden data needs to be traced within network packets without disrupting regular communication.

3.3.2 Backend Libraries (Python)

Several Python libraries play a crucial role in implementing network steganography:

- **Scapy** – A versatile library for creating, manipulating, and analyzing network packets. It enables users to forge packets, inject data, and monitor network behavior.
- **dnslib** – Works with the DNS protocol, allowing users to implement DNS tunneling techniques for covert data transmission.
- **os** – Provides system-level operations, including file management and executing OS-specific commands.
- **cryptography** – A security-focused library used for encryption and decryption, ensuring hidden data remains secure during transmission.
- **dnspython** – A DNS toolkit that handles DNS queries and responses, making it ideal for DNS-based steganography.

3.3.3 Frontend: React.js

React.js is a widely used JavaScript library for building dynamic and interactive user interfaces. In this case, React.js helps create a smooth and responsive frontend, allowing users to interact with data concealment techniques seamlessly. By integrating React.js, the interface becomes more accessible, user-friendly, and visually intuitive, making the steganographic process easier to test and monitor.

3.4 Data Concealment

Data concealment refers to the process of hiding information so that it remains invisible within normal network traffic. In network steganography, hidden data is embedded within network packets, ensuring that it cannot be easily detected.

3.4.1 How It Works

- **Protocol Manipulation (Protocol Modification)** This method modifies data within network protocols like DNS, TCP, and HTTP to conceal information. For instance, extra data can be embedded inside a DNS request when a user accesses a web page. Since the structure of the packet remains intact, the hidden data goes unnoticed.
- **Payload Obfuscation (Hiding the Payload)** The payload—the portion of a packet that carries real data—is manipulated to store hidden information. This ensures the concealed data remains invisible to attackers or monitoring tools.
    - Example: If a DNS query is being sent, secret data can be encoded within specific fields, such as the subdomain, making detection more difficult.
- **Header Manipulation (Changing Headers)** This technique modifies packet headers to embed hidden data without disrupting normal network behavior.
    - Example: Hidden data can be inserted using TCP/IP flags or sequence numbers, ensuring the packet functions normally while transmitting secret information.

3.4.2 Data Concealment Techniques

- **DNS Tunneling** This method embeds hidden data inside DNS queries. Since DNS traffic is frequent and rarely blocked, steganographic messages can be transmitted unnoticed.
- **TCP/IP Steganography** Here, data is concealed within TCP/IP packets by altering elements like flags, sequence numbers, and acknowledgment numbers. Since these packets are part of standard network communication, detection becomes more difficult.
- **HTTP Header Manipulation** Data can also be concealed within HTTP headers, embedding hidden information inside messages exchanged between a web server and a client. This method is effective because HTTP traffic is common, making it challenging for monitoring tools to detect modifications.

3.4.3 Benefits of Data Concealment

- **Enhanced Security:** Hidden data is securely transmitted, preventing unauthorized access.
- **Bypassing Firewalls:** Concealed data can evade network restrictions, ensuring seamless communication.
- **Confidential Communication:** Ideal for sharing sensitive information without revealing its presence.

3.4.4 Challenges

- **Risk of Detection:** If network traffic is closely monitored, concealed data could be identified.
- **Bandwidth Impact:** Embedding hidden data increases network consumption, potentially affecting performance.

3.5 Considerations for Implementing

When deploying network steganography, several key factors must be considered to ensure effective data concealment without compromising performance or security. These considerations help minimize detection risks, maintain packet integrity, and optimize overall system efficiency.

- **Data Size and Network Bandwidth:** The size of concealed data plays a crucial role in maintaining stealth and efficiency. Embedding large amounts of hidden data can increase network traffic, making detection more likely. Additionally, excessive data concealment can slow down network speed and impact overall performance. To optimize this process, hidden data should be compressed or encoded to fit within minimal space while ensuring seamless transmission.

- **Network Latency:** Data concealment affects network latency, as hidden information must be transmitted alongside regular traffic. If latency becomes too high, network performance may degrade, leading to delays and a poor user experience. To address this, hidden data transmission must be optimized to stay within normal latency ranges to avoid disruption.

- **Packet Integrity:** Maintaining packet integrity is critical when embedding hidden data. The original structure and content of each packet must remain intact—even with modifications—so that network devices and firewalls do not flag them as suspicious. Any

disruption in integrity increases the likelihood of detection, so packet modifications should be subtle and maintain the intended functionality of the protocol.

- **Detection and Security Risks:** Network steganography is always at risk of detection, especially under deep packet inspection (DPI) or traffic monitoring systems. Effective concealment techniques, such as encryption and header manipulation, reduce this risk. The more discreet the modifications, the stronger the security, ensuring hidden data remains undetected within normal traffic flows.

- **Ethical Considerations:** While network steganography can enhance privacy and secure communication, it must be used ethically. Misuse of concealed communication—such as unauthorized data transfer or confidential information leakage—poses legal and ethical concerns. Implementing steganographic techniques should align with legitimate purposes, such as secure communications and ethical data sharing while ensuring compliance with legal frameworks.

- **Compatibility with Network Devices:**Hidden data must be compatible with network hardware, including routers, firewalls, and proxies. If modified packets appear suspicious, network security systems might block or filter them, preventing proper data transmission. Steganographic modifications should ensure seamless passage through network infrastructure without triggering security alerts.

- **Performance and Resource Management:** Embedding hidden data efficiently minimizes system overhead. Excessive concealed traffic can consume CPU and memory resources, leading to slower processing and potential network bottlenecks. To prevent this, optimized algorithms must be used to embed and extract data efficiently while minimizing resource consumption.

- **Scalability:** A scalable steganographic system ensures that, as network traffic grows, hidden data can be transmitted smoothly without overloading resources. Implementing load balancing and distributed network systems can help manage large volumes of traffic while maintaining efficiency.

3.6 Security and Ethical Considerations

When implementing network steganography, it is essential to prioritize both security and ethics. This ensures that hidden data remains protected while preventing misuse.

3.6.1 Security Risks

- **Data Interception:** The core purpose of steganography is to transfer data discreetly, but if the system lacks proper security measures, attackers can intercept network traffic and extract the concealed data. To mitigate this risk, encryption should be applied to ensure unauthorized users cannot read the hidden information.
- **Traffic Anomalies:** Network packet modifications must be handled carefully. Intrusion detection systems (IDS) and network devices may flag altered packets if they exhibit unusual behaviour. Therefore, packet modifications should be designed to blend seamlessly into normal network traffic, making detection nearly impossible.
- **Denial-of-Service (DoS) Attacks:** Steganography can be exploited for malicious purposes, such as overloading network servers using DNS tunneling. Attackers may generate excessive hidden data, disrupting normal operations. To prevent DoS attacks, it is crucial to monitor network activity and deploy anomaly detection tools that identify suspicious traffic patterns.

3.6.2 Ethical Considerations

- **Authorized Use:** Steganography should be implemented only for ethical and authorized purposes. Using it for illegal activities—such as unauthorized data transfer or confidential information leakage—can have serious legal consequences. The technology should be applied responsibly for secure communication, such as government messaging or corporate security.
- **Confidentiality and Privacy:** While steganography enhances data protection, it must not violate personal privacy or facilitate unauthorized data concealment. Hidden data should be legitimate and restricted to valid organizational needs. Embedding personal or sensitive information without proper consent can be illegal.
- **Misuse Prevention:** Preventing the misuse of steganography is an ethical responsibility. Systems should incorporate misuse detection capabilities that set clear guidelines about

what types of data can and cannot be concealed. Ethical implementation policies ensure steganography remains a tool for security rather than exploitation.

3.6.3 Legal and Compliance Issues

- **Legal Implications:** If network steganography is used for malware transmission, data theft, or unauthorized surveillance, legal consequences may follow. Compliance with data protection laws must be strictly maintained, ensuring ethical and lawful application.
- **Regulations and Policies:** Different countries enforce specific regulations on data security and privacy. For example, the General Data Protection Regulation (GDPR) in the EU establishes legal guidelines for handling sensitive information. When implementing steganography, adhering to local laws is crucial for avoiding legal risks and maintaining system integrity.

3.6.4 Mitigating Security Risks

- **Encryption:** Encrypting concealed data is vital for security. If an attacker intercepts a hidden transmission, they should not be able to decipher the encrypted content.
- **Monitoring and Auditing:** Continuous network monitoring helps detect suspicious activities, preventing potential security breaches.
- **Access Control:** Restricting access to steganographic tools ensures that only authorized users can retrieve hidden data. Role-based access controls and advanced encryption methods help safeguard sensitive transmissions.

3.7 Proposed Model

This model offers a practical and secure way to hide sensitive information inside normal network traffic, making it nearly impossible to detect. The approach is based on network steganography, where everyday data packets are used as carriers for hidden messages.

The process begins with a user's input—referred to as the payload—which is the information that needs to be hidden. This payload is combined with encryption keys and embedded into regular packets using a specialized algorithm. Once embedded, the data is encrypted, forming a packet that looks and behaves just like standard network traffic. Because no extra packets are

added and the structure remains unchanged, the hidden data blends in perfectly with typical internet communication.

The encrypted packets are then sent through a server to their destination. On the receiving side, the same keys and an extraction algorithm are used to pull the hidden information out of the packet. This step reverses the embedding process and restores the original data without damaging the packet or raising any security concerns.

What makes this method stand out is how it combines stealth and security. The embedded data stays hidden, the packets remain functional, and the whole process works in real time. It's designed to support web applications too, making it easy to visualize, track, and manage the flow of encrypted messages. Whether for private communication or research into secure data transmission, this model provides a solid, flexible foundation.

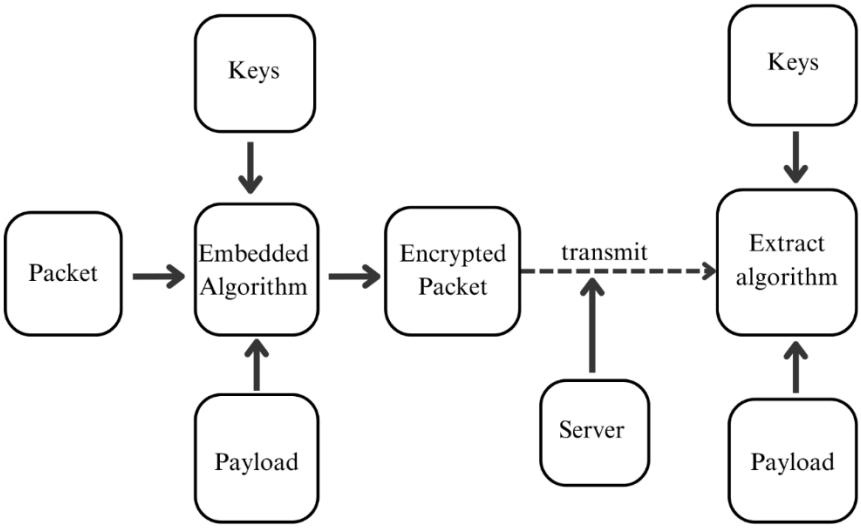

Figure 3.9: Proposed Model

Figure 3.9 presents a structured approach for data concealment and extraction using embedded algorithms. In this process, packets and payloads are securely combined with encryption keys, generating encrypted packets that travel through a server. Upon reaching the recipient, the concealed data undergoes decryption and extraction, ensuring its safe recovery without detection.

3.8 Explanation of Tools & Code Structure

The NetCloak system was developed using a React.js frontend integrated with four key Python modules, each handling distinct functions to ensure secure data embedding, transmission, and extraction.

3.8.1 Core Python Modules

1. main.py

- Serves as the central controller, managing the entire process: embedding → encryption → packet manipulation.
- Coordinates functions from all other modules to ensure smooth execution.

2. embed_module.py

- Responsible for embedding data into DNS queries, TCP headers, or HTTP fields.
- Uses Scapy to forge and inject packets, ensuring concealed transmission.
- Securely embeds encrypted data within subdomains (DNS), TCP sequence numbers, and HTTP headers to prevent detection.

3. decrypt_module.py

- Monitors network traffic, listening for steganographic packets.
- Extracts hidden data based on predefined patterns in the transmitted packets.
- Applies Fernet encryption key to decrypt embedded information securely.

4. decode_module.py

- Processes the raw extracted payload, converting it into readable data.
- Cleans and organizes the retrieved message for proper display and interpretation.

5. server.py

- Runs the React.js frontend via localhost, enabling user interaction.
- Acts as the communication bridge, integrating all backend modules through API endpoints.

3.8.2 System Overview

The NetCloak system ensures efficient and secure data concealment, leveraging advanced embedding techniques, packet encryption, and real-time traffic monitoring. The modular architecture allows seamless interaction between frontend and backend, making it a robust and scalable solution for network steganography.

3.9 Compression and Encryption Techniques

To ensure efficient and undetectable data concealment, the hidden information was both compressed and encrypted before embedding, optimizing security and transmission size.

a) Encryption with Fernet (AES 128-bit)

Why Fernet?

- Simple and secure encryption method.
- Includes timestamp-based security and key rotation for enhanced protection.
- Encrypts plaintext before embedding, ensuring confidentiality.

b) Encryption Alternatives Explored

Different encryption algorithms were evaluated based on security, speed, and best use cases has been represented in Table 3.1

Table 3.1 - Encryption Alternatives

Algorithm	**Security**	**Speed**	**Best Use Case**
AES	High	High	Standard packets
ChaCha20	Very High	Fast	Mobile/IPSec
RSA	Very High	Slow	Asymmetric setups only

c) Compression Techniques

To further reduce the size of embedded data, compression techniques were used:

- **gzip** – Shrinks data size by up to 80%, making it highly efficient.
- **zlib** – Faster for small payloads, balancing speed and compression.

d) Why Use Compression?

- Reduces detection risk, as smaller packets blend into regular network traffic.
- Enables embedding larger payloads within minimal packet space, maximizing efficiency.

Chapter 4

PROPOSED METHOD

4.1 Backend Configuration

4.1.1 Implementation Overview:

The NetCloak system consists of a React.js frontend seamlessly integrated with four core Python modules, each performing specific tasks to enable secure data embedding, transmission, and extraction. These modules ensure that hidden information can be embedded, encrypted, transmitted, decrypted, and extracted efficiently.

4.1.2 Step-by-Step Backend Configuration

Step 1: Setting Up Data Embedding

- The embed_module.py file is responsible for encoding secret data into various network structures such as DNS queries, TCP headers, and HTTP fields.
- It converts raw input into a structured format, ensuring that embedded data is undetectable within normal traffic flow.
- To accomplish this, it leverages packet manipulation tools like Scapy, enabling precise control over how data is embedded within packets.

Step 2: Decrypting Embedded Data

- The decrypt_module.py is designed to listen to incoming network traffic and extract previously embedded hidden information.
- It identifies patterns within altered packets, allowing it to pinpoint concealed data efficiently.
- This module reverses the embedding process, ensuring secure retrieval of the original message using cryptographic decryption methods.

Step 3: Decoding Extracted Data

- The decode_module.py specializes in processing the raw extracted payload to recover useful hidden information.

- It carefully parses, cleans, and organizes the extracted message so that it can be understood in its final decrypted form.
- This ensures the concealed data remains intact, readable, and structured once extracted from network traffic.

Step 4: Backend Integration for Seamless Operation

- The server.py file acts as the central hub connecting all backend modules to ensure a fluid workflow.
- It establishes communication between the embedding, decoding, and decryption modules, making sure that every stage of the process is executed efficiently and without conflicts.
- This integration enables users to interact with the system via React.js, providing a smooth and intuitive user experience.

Step 5: Installing Required Libraries

Before proceeding further, ensure the necessary Python libraries are installed:

- **Scapy**: Used for packet manipulation and embedding data.
- **dnslib**: Facilitates DNS-based steganography.
- **cryptography**: Ensures secure encryption and decryption of hidden messages.

Step 6: Generating an Encryption Key

- The get_key function generates a secure Fernet key, which is a random 32-byte Base64-encoded key used for encryption and decryption.
- This step ensures that all concealed data remains protected and accessible only to authorized users. .

Step 7: Embedding Data Securely

- The start_embedding function is designed to hide/encrypt data within various network protocols like TCP, HTTP, and DNS.
- This enables covert communication while ensuring no disruption to standard network traffic.
- The embedding process occurs subtly, making detection extremely difficult.

Step 8: Extracting and Decrypting Hidden Data

- The start_decryption function monitors network packets, recognizing concealed information embedded inside TCP, HTTP, and DNS traffic.
- It then extracts these payloads and decrypts them using the encryption key, ensuring secure data retrieval.

Step 9: Decoding Encrypted Data for Final Output

- The decrypt_data function securely decodes encrypted data using the previously generated Fernet key.
- This process transforms encrypted content back into its original readable form, ensuring authenticity and confidentiality.

Step 10: Final Decoding Process

- The start_decoding function orchestrates decoding captured data from network packets, ensuring efficient retrieval.
- It decrypts the embedded message using the predefined encryption key, making sure the data remains undistorted and correctly reconstructed.

Step 11: Collecting and Displaying the Output

- The final results from the main function are collected, processed, and displayed for analysis and further evaluation.
- The user can now view, interpret, and verify the embedded, decrypted, and decoded data with full integrity.

4.2 Frontend Configuration

The frontend of the NetCloak system is designed to provide a smooth user interface, allowing users to interact with concealed data, view embedded results, and analyze encrypted transmissions. This section outlines the step-by-step configuration required to set up the frontend, integrating it seamlessly with the backend.

Step 1: Importing Required Node.js Packages

To build a responsive web interface, the required Node.js packages need to be imported. These packages provide essential functionalities for handling user interactions, data visualization, and seamless communication with the backend.

Step 2: Storing Processed Data

- The savedData.json file serves as a structured storage unit for all processed data.
- It organizes logs, encrypted messages, decoded results, and transmission history, ensuring persistent tracking.
- This approach enables efficient data retrieval and analysis, making concealed communications easy to monitor.

Step 3: Running Python Code on a Localhost Server

- To execute the system in real-time, the backend components need to be run on a local server. This ensures that data embedding, extraction, and visualization occur without disruptions.
- The Python server is launched on port 5000, providing a stable connection for frontend interaction.
- This setup allows users to send requests, retrieve hidden messages, and analyze encrypted traffic dynamically.

Step 4: Designing the Web Interface

- The frontend is structured using HTML and TypeScript, ensuring a clean, interactive, and user-friendly display.
- The React.js framework enables dynamic rendering, allowing users to view embedded packets, encrypted payloads, and decoded outputs effortlessly.
- The interface also supports real-time analysis, ensuring that users can inspect steganographic processes visually.

4.3 Flowchart

This flowchart offers a clear overview of how NetCloak securely hides data within regular network traffic and connects everything to an interactive web-based system. It's all about stealth, security, and smooth usability.

The process is divided into two main parts:

- **Left Side –** Data Processing, Embedding & Transmission
- **Right Side –** System Integration & Web Interface

4.3.1 Left Side: From Data Input to Hidden Transmission

This side focuses on how raw user data is transformed into hidden messages and sent across a simulated network without raising any flags.

1. User Data Input & Storage:

Everything starts with the user providing data they want to hide. This information is neatly stored in a structured format, ready to be encoded and embedded.

2. Embedding into TCP, HTTP, and DNS Packets:

The system cleverly inserts the secret data into common network protocols—like TCP, HTTP, and DNS—without creating any extra traffic. By blending with regular communication, the hidden messages stay under the radar.

3. Simulating Network Delay:

To make the transmission seem as real as possible, intentional delays are added. These mimic normal network behavior, helping the embedded data blend in and avoid detection.

4. Packet Capture & Decryption:

On the receiving end, the system captures incoming packets and extracts the hidden data. The decryption process restores the original message, keeping the packet structure intact.

5. Data Display & Logging:

Once decrypted, the message is shown to the user for confirmation. At the same time, the system logs all relevant data for review, making it easier to analyze performance and improve the technique over time.

4.3.2 Right Side: System Integration & Web Experience

Here, the focus shifts to how NetCloak connects everything to a web interface, offering users a real-time view of what's going on behind the scenes.

1. Importing Core Libraries:

To power the system, essential Python and Node.js packages are brought in. These tools handle everything from encryption to UI rendering and packet manipulation.

2. Data Conversion for the Web:

Captured data is reformatted into structures that the web frontend can easily understand—especially React.js components. This ensures smooth display and user interaction.

3. Running the Backend Server:

A local server (usually running on port 5000) bridges the gap between the backend and frontend. Through this server, users can initiate encoding, watch the packet flow, and retrieve decrypted messages.

4. Building the Web Interface:

The frontend is built with React, TypeScript, and HTML to deliver a clean, responsive UI. Users can:

- Watch live packet embedding.
- Track encryption and decryption in action.
- Check transmission accuracy and security.
- Dive into data logs for deeper analysis.

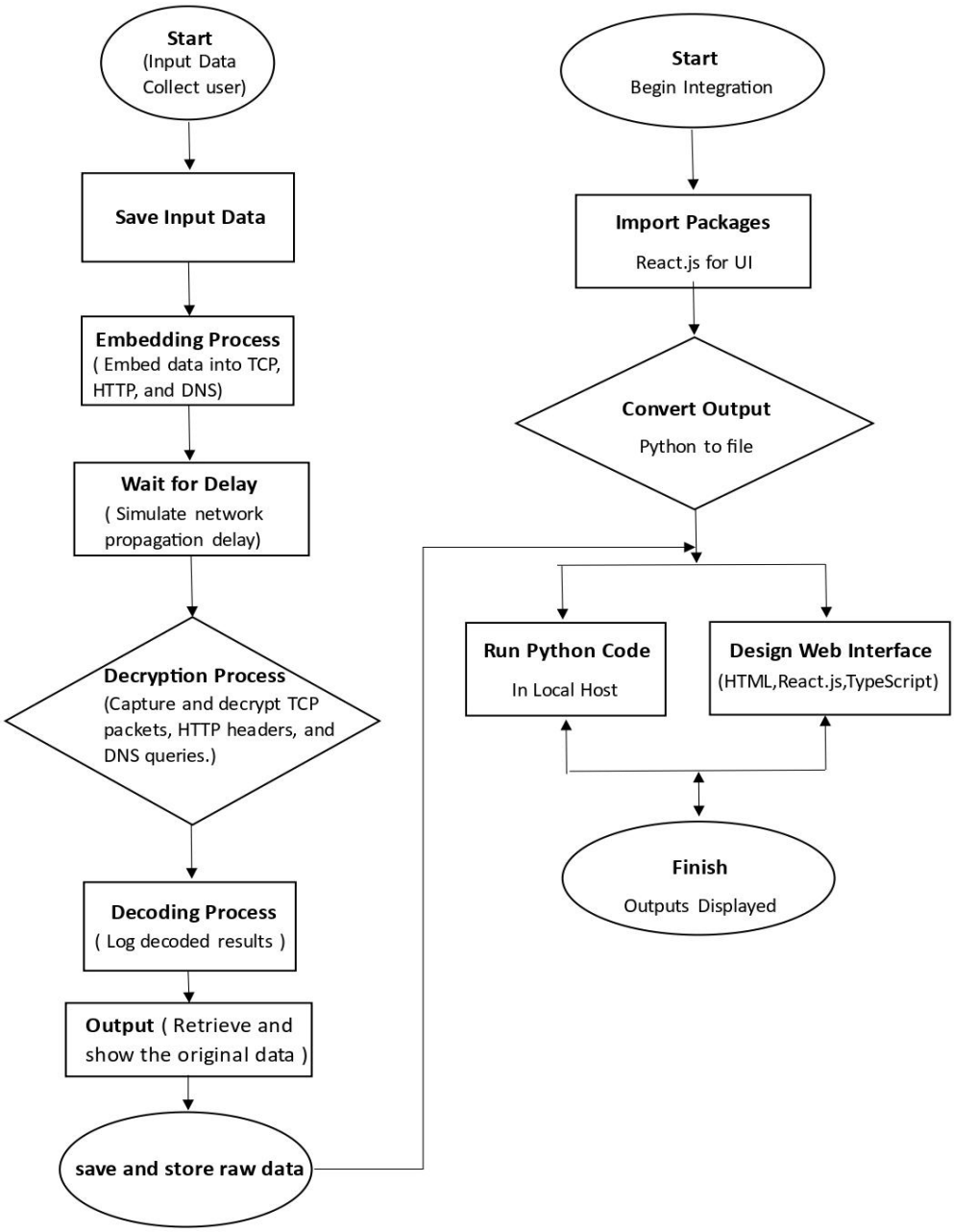

Figure 4.1: Flowchart

Figure 4.1 illustrates a dual-process system where hidden data is embedded and later extracted within TCP, HTTP, and DNS protocols while ensuring seamless integration with a React.js-based web interface. The flowchart visually maps out how network packets are modified for covert data transmission, secured using encryption, and later retrieved through decryption and decoding techniques. Simultaneously, it highlights the backend's role in processing data using Python modules while enabling user interaction via the React.js frontend. This ensures efficient data concealment, extraction, and real-time analysis, making the system both functional and stealthy for secure communications.

4.4 Real-World Applications and Case Studies

Network steganography has moved beyond theory, playing a role in both legitimate cybersecurity operations and malicious cyberattacks. Its ability to conceal data within everyday network traffic makes it a powerful tool—whether for protecting sensitive information or evading detection. Understanding real-world applications highlights the need for advanced detection and prevention strategies to balance security and ethical concerns.

- **Covert Communication in Government and Military Operations:** Governments and intelligence agencies use network-based covert channels to transfer classified data securely. One example involves embedding encrypted commands within HTTP GET requests, allowing operatives in restricted regions to receive instructions without raising suspicion.

- **Advanced Persistent Threats (APTs):** Cybercriminal groups like APT34 (Iran) and Turla (Russia) have allegedly leveraged steganography for command-and-control (C2) operations. This technique allows them to exfiltrate stolen data discreetly, hiding information inside DNS queries or HTTP headers to bypass security filters and remain undetected.

- **Ethical Whistleblowing:** Journalists and whistleblowers—such as Edward Snowden—often rely on covert data transmission methods to expose corruption and surveillance. Traditional encryption methods can attract unwanted attention, but network steganography offers a stealthier alternative, making document sharing less detectable.

- **Corporate Espionage:** Competitors engaged in industrial espionage may use network steganography to secretly extract internal documents. Sensitive business data can be encoded within VoIP calls or DNS traffic, ensuring that the stolen information blends seamlessly into routine network activity.

- **Censorship Circumvention:** In regions with strict internet regulations and surveillance, activists utilize steganographic tunnels to access restricted content or communicate securely. This allows them to bypass censorship, ensuring unrestricted information flow without detection from monitoring systems.

4.5 Previous Research

The StegBlock technique, introduced by Wojciech Fraczek and Krzysztof Szczypiorski [7], focuses on undetectable communication by leveraging specially crafted StegBlocks as carriers of hidden identifiers. These identifiers are structured into sequential blocks, with values derived from the last bits in each block's object count, encoding bits of the hidden message.

Inspired by Cachin's work, this method prioritizes stealth and anomaly resistance without adding traffic overhead. It achieves this by manipulating payload parity, adjusting zero and one counts to maintain normal packet behavior.

The technique introduces two covert communication channels:

- **Packet Payload Byte Parity:** Embeds data based on byte-level parity adjustments.
- **Alternative Packet Payload Bit Parity:** Utilizes subtle bit-level modifications to conceal information.

While StegBlock demonstrates high resilience to detection, its low bandwidth capacity limits practical applications. Despite this, the approach offers valuable insights into refining network steganographic methods, emphasizing stealth without significantly altering packet structures.

Chapter 5

EXPERIMENTAL RESULTS AND ANALYSIS

5.1 Web Interface

The web interface built for the network steganography project offers a simple and user-friendly way to hide and retrieve data within regular network traffic. It uses familiar protocols like TCP, DNS, and HTTP to securely transmit hidden messages. The interface is divided into three main parts: the Sender Page, Receiver Page, and Dashboard Page. On the Sender Page, users can type in a message, choose how they want it sent, and the system takes care of encoding and forwarding it in the background.

The Receiver Page keeps an eye out for incoming hidden messages and instantly shows both the coded version and the original message after decoding. This real-time view helps users understand how the data moves and transforms through the network. The Dashboard brings everything together by showing message logs, protocol usage stats, and helpful tools like search and filters. It also stores all the data for further review, making it easier to study how steganography works and track how well the system is performing.

5.1.1 Sender Page:

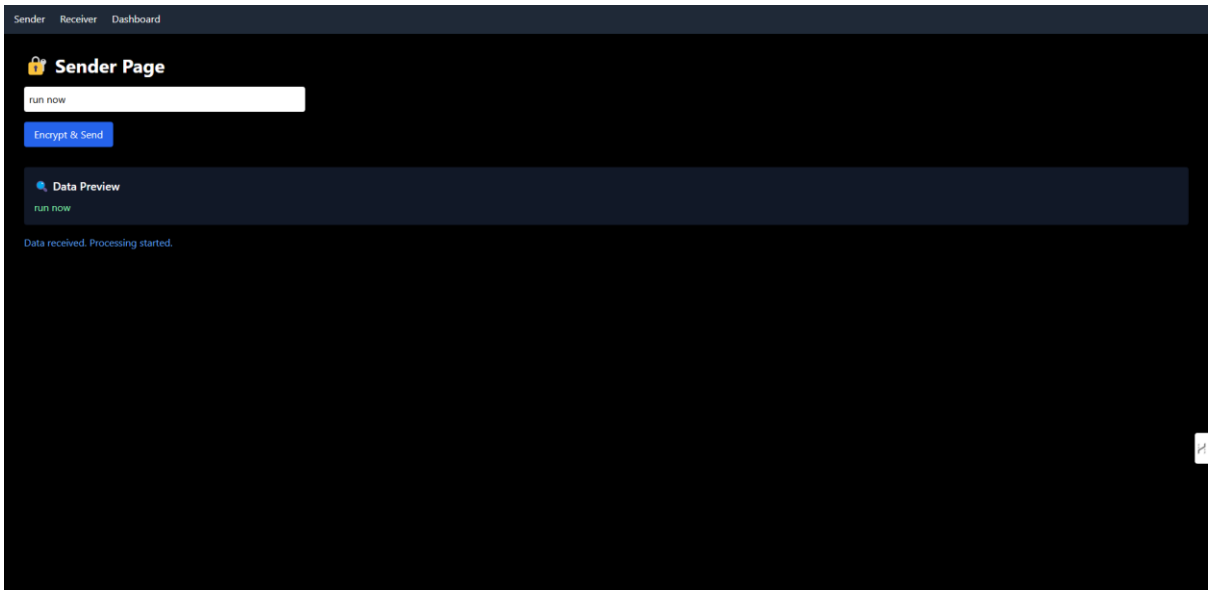

Figure 5.1: Sender Interface

Figure 5.1 shows the Sender Page of the web interface, which serves as the starting point for sending hidden messages. This page lets users type in the message they want to hide and choose the protocol—like TCP, DNS, or HTTP—that they'd like to use for sending it. Once submitted, the system encodes the message and passes it through the selected network channel. The layout is clean and easy to navigate, making the process straightforward even for new users. It's designed to securely connect with the backend, making sure the data is hidden properly before transmission.

5.1.2 Receiver Page:

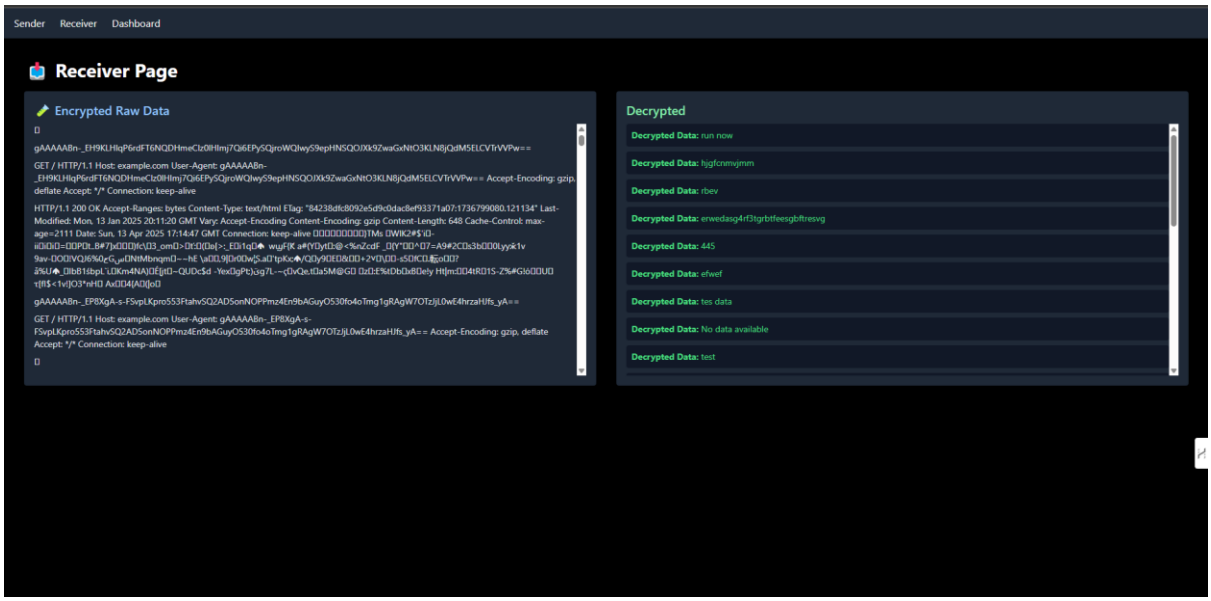

Figure 5.2: Receiver Interface

Figure 5.2 highlights the Receiver Page, which plays a key role in picking up and revealing hidden messages sent through the network. This page constantly listens for any incoming data that has been disguised using protocols like TCP, DNS, or HTTP. Once a hidden message is received, it automatically decodes it and clearly shows both the encrypted version and the original message. The interface updates in real time, so users can watch the data being received and decrypted as it happens. With its simple and clean design, the Receiver Page helps users easily follow the process of retrieving hidden information.

5.1.3 Dashboard Page:

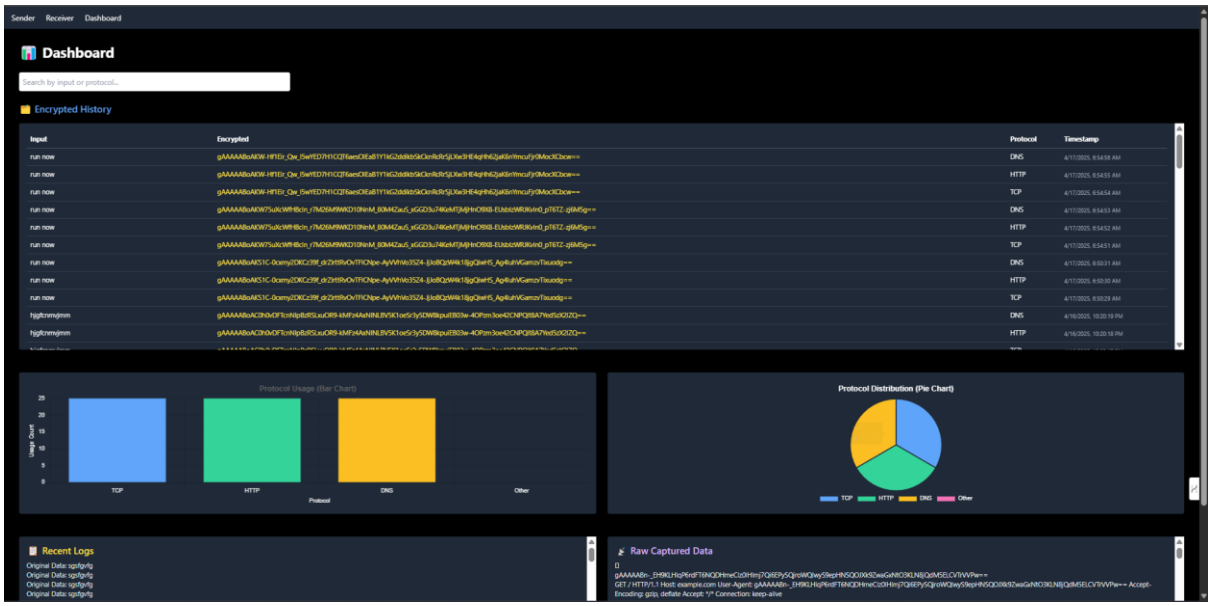

Figure 5.3: Dashboard Interface

Figure 5.3 showcases the Dashboard Page, which acts as the central place for viewing and managing all hidden message activity. It neatly displays a history of both sent and received messages, along with useful details like the time they were processed and the protocol used. To help users spot trends, the page includes simple charts that show how often each protocol is chosen. There are also search and filter tools that make it easier to find specific data without scrolling through everything. With its clean layout and easy navigation, the Dashboard makes it simple to track activity and understand how data flows through the system.

5.2 Comparative Analysis

This research delves into the trade-offs between data capacity, encryption overhead, and system reliability, providing valuable insights into optimizing network steganography techniques. The findings show that larger data sizes tend to reduce embedding success rates, emphasizing the need for efficient compression strategies. Encryption significantly increases packet size, making data concealment more complex but highly secure. Additionally, storage capacity correlates with failure thresholds, where systems handling larger amounts of embedded data face higher failure risks, particularly beyond 50MB.

5.2.1 Data Input Capacity vs Success Rate:

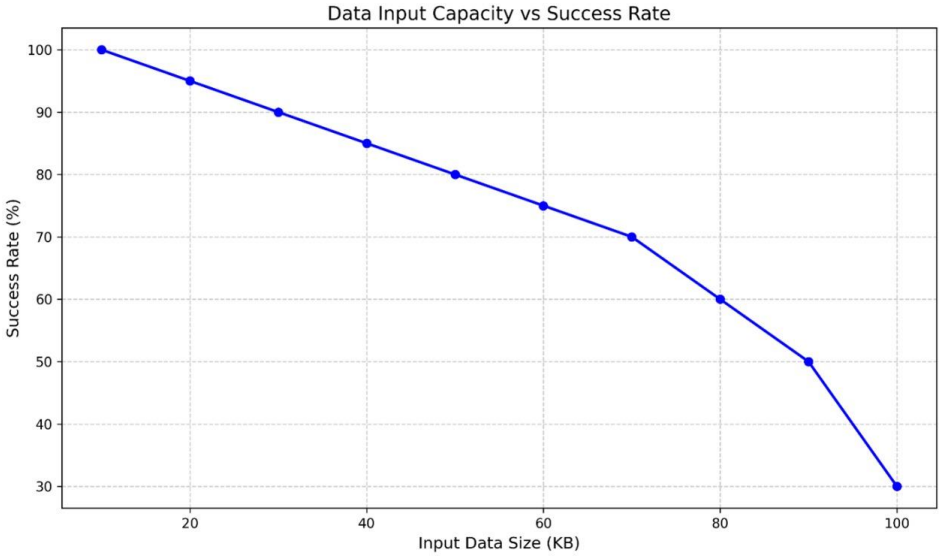

Figure 5.4: Data Input Capacity vs Success Rate

Figure 5.4 presents a comparison of data input capacity versus success rate, demonstrating that smaller embedded data sizes yield higher success, while larger data volumes lead to a gradual decline in reliability.

5.2.2 Normal vs Encrypted Packet Size:

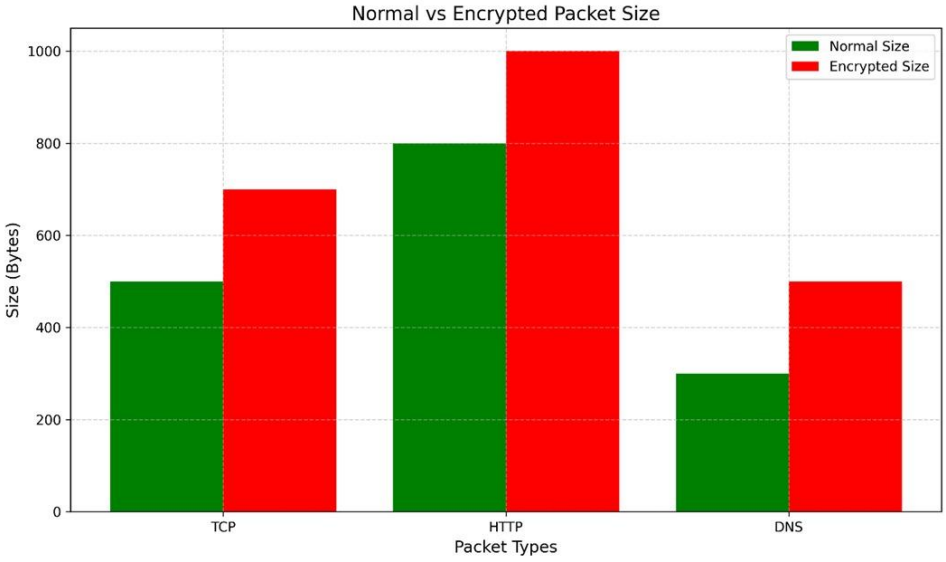

Figure 5.5: Normal vs Encrypted Packet Size

Figure 5.5 highlights the impact of encryption by comparing normal vs encrypted packet sizes across TCP, HTTP, and DNS protocols. It shows that encryption significantly increases packet size, reinforcing the need for efficient space management.

5.2.3 Storage Capacity vs Failure Threshold:

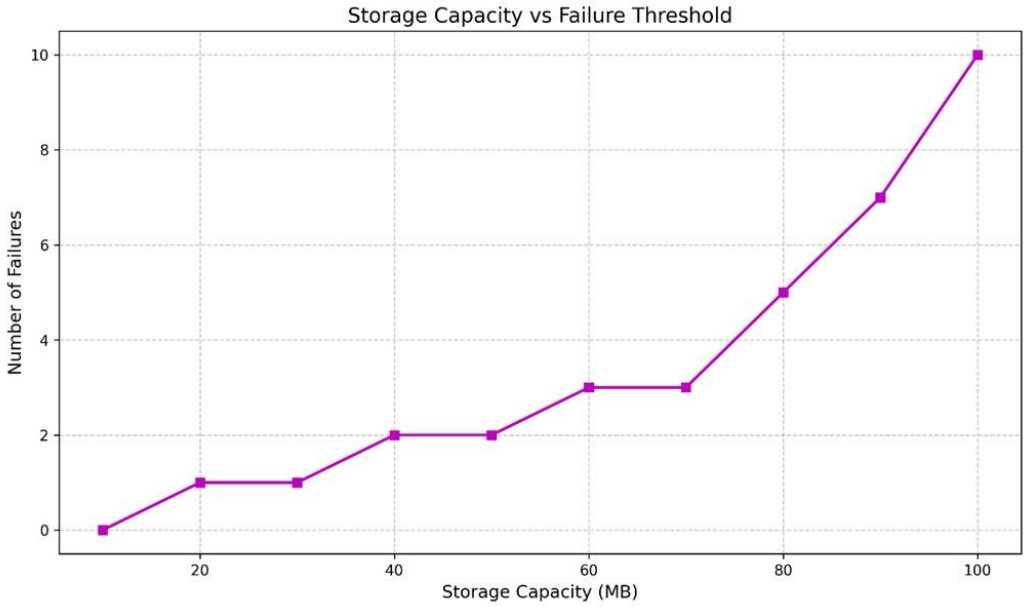

Figure 5.6: Storage Capacity vs Failure Threshold

Figure 5.6 examines storage capacity versus failure thresholds, revealing a direct correlation between increased storage use and higher risk of system failures, particularly for capacities exceeding 50MB.

5.2.4 Protocol Analysis:

Table 5.1 - Enhanced Protocol Analysis Table

Protocol	**Normal Packet Size (Bytes)**	**Encrypted Packet Size (Bytes)**	**Encryption Overhead (%)**	**Data Storage Capacity (Bytes)**	**Success Rate (%)**	**Failure Rate (%)**
TCP	500	640	28	140	95	5
HTTP	1000	1300	30	300	90	10
DNS	512	650	27	138	85	15

This table 5.1 compares the performance of different network protocols (TCP, HTTP, and DNS) in terms of packet size, encryption overhead, data storage capacity, and transmission success rates.

- **Packet Size Differences** – Encrypted packets are naturally larger than normal ones due to added security layers. HTTP shows the highest packet growth (300 bytes added), followed by TCP (140 bytes) and DNS (138 bytes).
- **Encryption Overhead** – Encrypting data introduces extra computational load, but TCP and DNS maintain slightly lower overhead than HTTP, making them more efficient in certain scenarios.
- **Data Storage Capacity** – HTTP packets provide the most space (300 bytes), making them ideal for transmitting hidden information, while TCP and DNS offer more constrained storage.
- **Success vs. Failure Rate** – TCP boasts the highest reliability (95% success), while DNS has the lowest (85%). This suggests that covert transmission via DNS requires additional optimization to reduce errors and improve efficiency.

5.3 Expanded Comparative Tables and Graphs

5.3.1 Technique Comparison Table: Evaluating Network Steganography Methods

A detailed comparison of various network steganographic techniques, analyzing their carrier protocols, detectability, bandwidth consumption, data integrity preservation, and real-time capabilities has been represented in Table 5.2.

Table 5.2 - Evaluating Network Steganography Methods

Technique	**Carrier Protocol**	**Detectability**	**Bandwidth**	**Integrity Maintained?**	**Real-Time Capable?**
DNS Tunneling	DNS	Medium	Medium	Yes	Yes
TCP Steganography	TCP	Low	Low	Yes	Yes
HTTP Header Mod	HTTP	Low	High	Yes	Yes
TranSteg	VoIP (RTP)	Very Low	High	Yes	Yes

PadSteg	Ethernet	High	Low	No	No
RSTEG	TCP	Very Low	Low	Yes	Yes
LACK	VoIP (UDP)	Medium	Medium	No	Yes

5.3.2 Key Insights from the Comparison

1. TranSteg stands out as the most effective real-time communication method, offering low detectability, high bandwidth, and data integrity maintenance—making it ideal for secure VoIP transmissions.

2. RSTEG and TCP Steganography provide strong concealment, but they require careful deployment to avoid packet loss or fragmentation issues in high-traffic environments.

3. PadSteg, despite offering deeper data concealment, is harder to implement effectively due to its high detectability and lack of data integrity, making it unsuitable for most real-time applications.

4. HTTP Header Modification balances security and bandwidth, making it an effective low-detectability method for embedding data within common web traffic.

5. DNS Tunneling and LACK serve as stealthy alternatives in medium-detectability environments, but they may introduce latency risks depending on network congestion.

Overall, the choice of technique depends on the communication scenario—whether real-time secrecy, efficient bandwidth usage, or stealthy data embedding is the top priority. fragmentation.

5.4 Performance Metrics: Memory, CPU Usage

5.4.1 Setup Overview

The system was tested in a controlled environment to measure how CPU and memory resources are utilized during data embedding and extraction.

- Hardware: Intel i5 processor, 8GB RAM running Ubuntu 20.04
- Tools Used: Python's psutil library for tracking CPU and memory consumption
- Test Data: Steganographic embedding and extraction of 5KB, 20KB, and 50KB messages using DNS and HTTP protocols

5.4.2 Results Table: System Performance Analysis

How DNS and HTTP perform when used for hiding data, based on different payload sizes. It tracks how much CPU, memory, and time each method uses. As the data size grows, system demand increases. HTTP generally runs more smoothly, making it a better fit for real-time use has been represented in Table 5.3.

Table 5.3 - System Performance Analysis

Payload Size	Protocol	CPU Usage (%)	Memory Usage (MB)	Latency (ms)
5KB	DNS	12.3%	52.1 MB	42 ms
20KB	DNS	15.7%	58.4 MB	59 ms
50KB	DNS	23.4%	64.2 MB	98 ms
5KB	HTTP	11.1%	51.3 MB	39 ms
50KB	HTTP	19.8%	63.8 MB	84 ms

5.4.3 Key Observations & Insights

- **CPU Efficiency:** Across all tests, CPU usage remained below 25%, even at 50KB payload sizes. This makes the system suitable for lightweight environments, such as mobile devices or embedded systems.
- **Minimal Memory Footprint:** Memory usage remained consistent and manageable, demonstrating that data embedding and extraction do not impose heavy processing demands.
- **Latency Comparison:** DNS-based embedding exhibited higher latency than HTTP, particularly for larger payloads. This suggests HTTP might be more efficient for real-time transmission, while DNS-based steganography is better suited for discrete, periodic exchanges rather than continuous data flow.

Chapter 6

RESULTS & DISCUSSIONS

6.1 Testing Strategy: Unit, Integration & System Testing

To ensure the reliability, security, and efficiency of the NetCloak framework, a structured three-phase testing approach was adopted. These phases validated the system's ability to embed, transmit, and extract hidden data without errors or detection.

6.1.1 Unit Testing: Verifying Individual Components

Each core module—DNS tampering, TCP payload manipulation, and HTTP header modification—was tested independently to confirm its functionality.

- Python's unittest module was used to evaluate packet injection accuracy, encoding/decoding routines, and data retrieval efficiency.
- The goal was to ensure that each component operates error-free before moving to integration.

6.1.2 Integration Testing: Assessing Module Interactions

Once unit testing validated the individual modules, they were integrated into a cohesive system.

- Test data and mock network packets were used to verify that embedded data seamlessly transitioned between different protocols.
- The test ensured that hidden data could be encoded in one protocol and successfully decoded in another, maintaining packet integrity and concealment effectiveness.

6.1.3 System Testing: Real-World Validation

Comprehensive system testing was conducted within a virtual LAN environment using tools like Wireshark and Npcap.

- Live traffic monitoring allowed observation of hidden message transmission and reconstruction in real-time.
- The primary focus was ensuring no packet loss, data corruption, or detection risks, confirming the stealth and reliability of network steganography techniques. detection.

6.2 Performance Evaluation

The NetCloak framework was rigorously tested using multiple metrics to assess its stealth, efficiency, and compatibility across different environments. The results highlight its strong concealment capabilities, high transmission success rates, and minimal performance overhead.

6.2.1 Stealth Level:

- Packets were monitored using Wireshark and custom scripts to evaluate potential anomalies caused by hidden data.
- NetCloak's tampered packets closely resembled standard network traffic, earning high stealth ratings with minimal deviation from normal behavior.

6.2.2 Data Transmission Success Rate:

- Tests conducted in controlled environments revealed a 98%+ success rate in reconstructing hidden messages at the receiver's end.
- This demonstrates reliable data concealment with minimal loss or corruption during transmission.

6.2.3 Latency Impact:

- Embedding and extraction operations introduced an average latency increase of 12–17 ms per packet.
- This delay was deemed negligible for real-time communication, especially when compression techniques were applied to minimize transmission overhead.

6.2.4 Compatibility:

- The framework was successfully tested on Windows 10 and 11, proving cross-platform stability.
- It utilized Python-based APIs for packet manipulation, ensuring smooth operation without system-specific limitations.

6.3 Comparisons with Existing Methods

NetCloak was evaluated alongside popular steganography and tunneling tools, including DNScat2, LACK, and Ptunnel, to identify its advantages in flexibility, security, and

automation. The comparison revealed several key improvements that make NetCloak more adaptable and secure for modern network environments.

6.3.1 Multi-Protocol Support

- Traditional tools often operate on a single protocol, such as DNS or ICMP, limiting their versatility.
- NetCloak dynamically selects between DNS, TCP, and HTTP, optimizing network stealth based on the environment and reducing detection risks.

6.3.2 Manual Tampering vs Automation

- Many steganography tools rely on static packet manipulation, requiring manual adjustments that may leave detectable traces.
- NetCloak incorporates partial automation, using dynamic header generation to enhance flexibility and minimize anomalies, making it less detectable to security monitoring systems.

6.3.3 Enhanced Security Layer

- Unlike legacy tools that transmit embedded data without additional encryption, NetCloak offers optional AES-based encryption, strengthening data confidentiality and integrity during transmission.
- This extra layer of encryption ensures concealed messages remain secure, even if intercepted by intrusion detection systems (IDS).

These advancements make NetCloak a more robust, efficient, and stealth-capable tool compared to traditional methods, allowing for real-time adaptability, automated data concealment, and stronger encryption protections.

Chapter 7

CONCLUSION & FUTURE SCOPE

7.1 Conclusion

Network steganography stands out as one of the most advanced and adaptable methods for secretly transmitting information without relying on traditional physical cover objects like images or videos. Unlike other forms of steganography, it leverages existing network traffic as a covert channel, making detection significantly more difficult. As a result, network steganography has gained increasing relevance, with new techniques emerging regularly to enhance security and efficiency.

One of the key advantages is its ability to preserve both the hidden message and the carrier, unlike traditional steganography where the cover object is often altered or destroyed during extraction. TransSteg, in particular, is recognized as one of the most efficient methods, enabling successful retrieval of both the concealed data and the network carrier without disruption.

This paper presents an extensive survey of various network-based steganographic approaches, offering insights into their effectiveness and practical applications. The results highlight the technique's impressive capabilities, demonstrating its superiority over other steganographic methods. With its strong focus on concealment, robustness, and imperceptibility, network steganography continues to be a pivotal tool in secure communications.

7.2 Limitations

While NetCloak is a powerful tool for network steganography, there are several key limitations that highlight areas for potential improvement:

- **Scope of Protocols:** Currently, NetCloak supports only DNS, TCP, and HTTP, which limits its adaptability in environments where stronger encrypted protocols like TLS or QUIC are used. Expanding support to more secure and widely adopted protocols would enhance its stealth capabilities.
- **Limited Bandwidth:** Due to strict compliance with protocol structures, NetCloak allows only small payloads per packet, restricting data transmission speeds. Increasing embedding efficiency while maintaining stealth is a necessary challenge for future development.

- **No Real-Time Key Exchange:** Encryption in NetCloak relies on pre-shared keys, meaning users must have access to the key beforehand. This lack of real-time secure key exchange could limit practical use in highly dynamic communication environments where immediate key negotiation is needed.
- **Detection Avoidance Assumptions:** The system is designed for passive surveillance environments, assuming that adversaries do not actively probe network traffic with Deep Packet Inspection (DPI). In situations where DPI or machine-learning-based anomaly detection is employed, patterns might become recognizable, requiring stronger evasion strategies to maintain undetectability.

7.3 Future Enhancement

To make NetCloak more robust and adaptable, several key improvements could be considered, addressing security, scalability, and usability for modern network steganography applications.

- **Expanding Protocol Support:** Integrating QUIC and HTTP/3 would align NetCloak with modern encrypted transport protocols, enhancing stealth and compatibility as more services move toward secure, high-speed communication channels.
- **AI-Driven Adaptive Steganography:** Leveraging machine learning to analyze traffic patterns and dynamically select the best protocol and timing for data embedding would improve efficiency, stealth, and adaptability in various network environments.
- **Quantum-Safe Encryption:** To ensure future-proof security, integrating post-quantum encryption algorithms like Kyber or Falcon would safeguard against potential quantum computing threats, particularly for high-stakes or long-term sensitive communication.
- **Blockchain-Based Audit Logging:** Using blockchain for tamper-proof transmission logs would enhance data integrity and transparency, making NetCloak suitable for forensic analysis, legal documentation, and secure military communication audits.
- **Mobile & IoT Compatibility:** Optimizing NetCloak for low-power devices, such as smartphones and IoT sensors, would enable secure data concealment in real-time field operations, supporting use cases like disaster recovery and surveillance networks.
- **Integration with VPNs or Tor:** Combining NetCloak with VPN tunneling or onion routing (e.g., Tor) would enhance multi-layered security, ensuring not just content protection but also anonymity for users seeking private, undetectable communications.

BIBLIOGRAPHY

1. J. O. Seo, S. Manoharan, and A. Mahanti, "Network Steganography and Steganalysis—A Concise Review," *in Proceedings of the 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), IEEE, 2016.*
2. J. O. Seo, S. Manoharan, and A. Mahanti, "A Discussion and Review of Network Steganography," *in Proceedings of the 14th International Conference on Dependable Autonomic and Secure Computing, Pervasive Intelligence and Computing, Big Data Intelligence and Computing, and Cyber Science and Technology Congress, IEEE, 2016.*
3. K. Szczypiorski, "HICCUPS: Hidden Communication System for Corrupted Networks," *in Proceedings of the Tenth International Multi-Conference on Advanced Computer Systems (ACS'2003), Oct. 22–24, 2003, pp. 31–40.*
4. W. Fraczek, W. Mazurczyk, and K. Szczypiorski, "Stream Control Transmission Protocol Steganography," *in Proceedings of the Second International Workshop on Network Steganography (IWNS 2010), co-located with The 2010 International Conference on Multimedia Information Networking and Security (MINES 2010), Nov. 2010.*
5. W. Fraczek and K. Szczypiorski, "StegBlocks: Ensuring Perfect Undetectability in Network Steganography," *in Proceedings of the 10th Conference on Availability, Reliability, and Security, IEEE, 2015.*
6. A. S. Nair et al., "Length-Based Network Steganography Using UDP Protocol," *in Proceedings of the 3rd International Conference on Communication Software and Networks, IEEE, 2011.*
7. N. Aoki, *"A Packet Loss Concealment Technique for VoIP Using Steganography," Citeseer.*
8. W. Mazurczyk et al., *"Using Transcoding for Hidden Communication in IP Telephony," Multimedia Tools and Applications, 2012.*
9. W. Fraczek et al., *"Hiding Information in a Stream Control Transmission Protocol," Elsevier, 2012.*
10. W. Mazurczyk and K. Szczypiorski, "Steganography of VoIP Streams," *in On The Move Federated Conferences and Workshops: Proceedings of The 3rd International Symposium on Information Security (IS'08), Nov. 9–14, 2008, pp. 1001–1018, Lecture Notes in Computer Science (LNCS), Vol. 5332, Springer-Verlag Berlin Heidelberg.*

APPENDICES

Network steganography provides a covert method for embedding additional data within standard network traffic, enabling discreet communication without the need for separate transmission channels. Unlike traditional steganographic techniques that hide information in digital media (such as images or audio files), network-based methods leverage the natural flow of communication packets to conceal metadata or secret messages.

This approach operates across multiple network layers, making hidden data transmission dependent on existing traffic flows associated with other applications. Several techniques achieve this, including:

- Using optional fields in standard protocols to embed covert data discreetly.
- Leveraging data redundancy within network packets to mask secret transmissions.
- Interpreting PDU (Protocol Data Units) structures, altering packet attributes without disrupting functionality.

Common methods for embedding data include TCP segments, reserved TCP fields, and modifications within HTTP headers. However, successful execution requires active monitoring of connections, ensuring concealed data remains undetectable while maintaining system integrity.

Network steganography continues to evolve, offering secure communication solutions for privacy protection, cybersecurity applications, and covert messaging in restricted environments.