PROJECT REPORT ON

Project Report: Privacy Detector Chat System

SUBMITTED TO MIT SCHOOL OF COMPUTING, LONI, PUNE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE

BACHELOR OF TECHNOLOGY

(INFORMATION TECHNOLOGY)

Ayush Athare	MITU23BTITD003
Karthik Reddy	MITU22BTIT0040
Mahima Kela	MITU22BTIT0044
Parth Datar	MITU22BTIT0056

Prof. Kalyani Lokhande

Abstract

The project “Privacy Detector Chat System” focuses on developing a Python-based chat module capable of detecting and preventing privacy violations in user messages. The system scans chat inputs in real time to identify sensitive information such as email addresses, phone numbers, credit card numbers, social security numbers (SSNs), and URLs. It then sanitizes or blocks such messages to ensure data confidentiality and prevent accidental disclosure. The project combines regular expressions, database storage, and logging techniques to create a secure and efficient chat monitoring environment.

Aim

To design and implement an automated chat privacy detector that identifies and handles sensitive or personally identifiable information (PII) in user-generated messages, ensuring secure communication and data integrity.

Methodology

1. **Pattern Recognition using Regular Expressions (Regex):**
   Specific regex patterns were defined to detect privacy-sensitive information such as emails, phone numbers, credit card details, SSNs, and URLs.
2. **Database Implementation (SQLite):**
   All chat messages, including flagged ones, are stored in an SQLite database (chat_messages.db) for recordkeeping and auditing purposes. Each message entry records:
   - Timestamp
   - Raw message
   - Sanitized message
   - Privacy flag
   - Type of privacy issue detected
3. **Logging Mechanism:**
   The system maintains a log file (privacy_alerts.log) to record detected violations with timestamps for monitoring and review.
4. **Sanitization and Output:**
   The system sanitizes user inputs using HTML escaping to prevent cross-site scripting (XSS) and displays warnings when a privacy violation is detected.
5. **Interactive Chat Loop:**
   The program runs an interactive chat session where users can input messages. Each message is analyzed, sanitized, logged, and stored.

Results

The implemented system successfully:

- Detected multiple types of privacy violations (emails, phone numbers, URLs, etc.).
- Logged each violation accurately in the log file with proper timestamps.
- Stored both flagged and safe messages in the database.
- Prevented display of unsafe content by sanitizing it before output.
- Provided a functional, real-time privacy monitoring tool that ensures safer chat interactions.

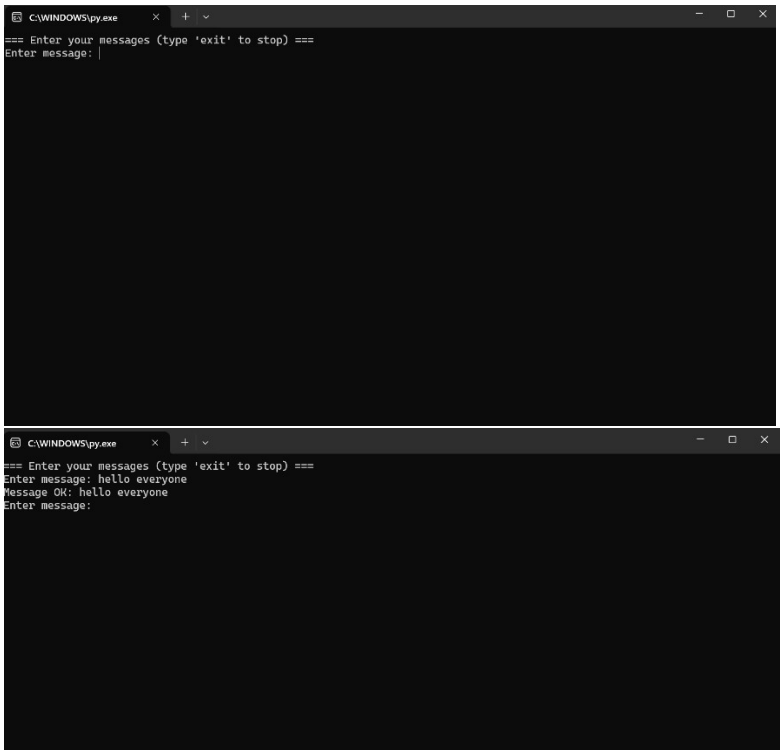
Code:-

```
#!/usr/bin/env python3
"""
privacy_detector_chat.py
Detects potential privacy violations in chat messages:
- Personal info (emails, phone numbers, SSN-like numbers)
- Credit card numbers
```

```
- Addresses (basic pattern)
- URL sharing
- Suspicious/excessively sensitive messages

Safe: messages are sanitized, logged, and optionally blocked.
"""

import re
import html
import sqlite3
import logging
import time

# ---------- Configuration ----------
MAX_MSG_LENGTH = 1000
DB_FILE = "chat_messages.db"
SUSPICIOUS_LOG = "privacy_alerts.log"

# ---------- Logging setup ----------
logging.basicConfig(level=logging.INFO)
s_logger = logging.getLogger("privacy_alerts")
s_handler = logging.FileHandler(SUSPICIOUS_LOG)
s_handler.setFormatter(logging.Formatter("%(asctime)s: %(message)s"))
s_logger.addHandler(s_handler)
s_logger.propagate = False

# ---------- Privacy Patterns ----------
PRIVACY_PATTERNS = {
    "Email": re.compile(r"[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+"),
    "Phone": re.compile(r"\b\d{10,15}\b"),  # basic international/local phone numbers
    "CreditCard": re.compile(r"\b(?:\d[ -]*?){13,16}\b"),
    "SSN": re.compile(r"\b\d{3}-\d{2}-\d{4}\b"),
    "URL": re.compile(r"https?://[^\s]+"),
    # Add more patterns if needed (addresses, passport, bank info)
}

# ---------- Database ----------
def init_db(conn: sqlite3.Connection):
    cur = conn.cursor()
    cur.execute("""
        CREATE TABLE IF NOT EXISTS messages (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            ts INTEGER NOT NULL,
            raw_text TEXT NOT NULL,
            sanitized_text TEXT NOT NULL,
            privacy_flag INTEGER NOT NULL,
            privacy_type TEXT
        )
    """)
    conn.commit()

def store_message(conn, raw_text, sanitized_text, flag, ptype=None):
    ts = int(time.time())
    cur = conn.cursor()
    cur.execute(
```

```
        "INSERT INTO messages (ts, raw_text, sanitized_text, privacy_flag, privacy_type) VALUES (?, ?, ?, ?, ?)",
        (ts, raw_text, sanitized_text, 1 if flag else 0, ptype)
    )
    conn.commit()


# ---------- Detector ----------
def detect_privacy_violation(msg: str):
    """Returns (flag: bool, violation_type: str) if privacy info detected."""
    for name, pattern in PRIVACY_PATTERNS.items():
        if pattern.search(msg):
            return True, name
    return False, None


def sanitize_for_display(msg: str):
    return html.escape(msg)


# ---------- Main Chat Loop ----------
def chat_loop():
    conn = sqlite3.connect(DB_FILE)
    init_db(conn)


    print("=== Enter your messages (type 'exit' to stop) ===")
    while True:
        msg = input("Enter message: ").strip()
        if msg.lower() == "exit":
            break
        if not msg:
            print("Empty message ignored.")
            continue
        if len(msg) > MAX_MSG_LENGTH:
            print(f"Message too long (max {MAX_MSG_LENGTH} chars).")
            continue


        # Detect privacy violation
        flag, ptype = detect_privacy_violation(msg)
        sanitized = sanitize_for_display(msg)


        if flag:
            s_logger.warning(f"Privacy violation detected: {ptype} | Msg: {msg}")
            print(f" ⚠️ Privacy violation detected ({ptype}). Message blocked or sanitized.")
        else:
            print(f"Message OK: {sanitized}")


        # Store in DB (even flagged ones, for audit/logging)
        store_message(conn, msg, sanitized, flag, ptype)


    conn.close()
    print("Chat session ended.")


if __name__ == "__main__":
    chat_loop()
```

Output:-



Conclusion

The Privacy Detector Chat System effectively demonstrates the ability to protect users from unintended data leaks during chat interactions. By integrating regex-based detection, logging, and data sanitization, it ensures that privacy-sensitive information is identified and handled securely. The project proves valuable for chat applications, customer support bots, and internal communication tools requiring confidentiality.

Recommendations

- **Enhance Pattern Coverage:** Add more comprehensive patterns to detect passport numbers, bank details, and addresses.
- **Implement Encryption:** Encrypt stored messages in the database for better security.
- **Introduce a GUI:** Develop a user-friendly graphical interface to enhance usability.
- **Integrate with Web Platforms:** Deploy the module as a backend component in real-world chat or customer service applications.
- **Machine Learning Extension:** Use NLP and ML models to identify context-based privacy risks beyond regex detection.