Remote Keylogger with Web-Based Server Control

Overview

This enhanced keylogger system is designed for remote deployment scenarios where the keylogger client sends data to a persistent web-hosted server. The server now stores all collected data **in-memory** and provides a comprehensive web-based control panel for managing and accessing this data. This setup allows for flexible data collection and remote management.

Architecture

Client-Server Model

- **Keylogger Client (`keylogger.py`)**: Runs on target machines, captures data (keystrokes, mouse events, screenshots), and sends it to the remote server. It also includes logic to **automatically start the `server.py` if it's not already running locally**.
- **Remote Server (`server.py`)**: A Flask-based application designed to be hosted on a web platform. It receives and stores all keylogger data **in-memory**. It provides a web interface for authenticated users to view, list, and download collected data. **Data stored in memory will be lost upon server restart.**
- **Web Control Panel**: A browser-based interface served by `server.py` that allows authenticated users to interact with the collected data and manage the server.

Key Features

1. **Remote Data Collection**: All captured data is sent to a centralized remote server.
2. **In-Memory Data Storage**: All logs, encryption keys, and screenshots are stored directly in the server's memory for quick access. **Note: Data is not persistent and will be lost if the server process is terminated or restarted.**
3. **Web-Based Control Panel**: Provides a user-friendly web interface for:
4. **Login**: Basic username/password authentication to access collected data.
5. **Session Listing**: View a list of all uploaded sessions.
6. **Session Viewing**: Inspect the files within each session.
7. **File Download**: Download individual log files, encryption keys, or screenshots.

8. **Bulk Download**: Download all collected data as a single ZIP archive.
9. **Automatic Server Start (Local)**: The `keylogger.py` script will automatically attempt to start `server.py` in the background if it detects that the server is not running on the configured `UPLOAD_URL` (typically `127.0.0.1:5000` for local testing).
10. **Built-in Decryptor**: A `decryptor.py` script is automatically included with each session download from the server, simplifying the decryption process for collected logs.

Installation and Deployment

Server Deployment (`server.py`)

Prerequisites

```
# Required Python packages
pip install flask cryptography
```

Webhosting Deployment Steps

1. **Upload Server File**:
   Upload the `server.py` file to your webhosting environment. This single file contains all the necessary server logic and the embedded web interface.

2. **Configure Server**:

3. Ensure `server.py` is set as the entry point for your Flask application.
4. Verify port configuration (default: 5000).
5. For production, ensure `debug=False` is set within `server.py` (it is by default in the provided `server.py`).

6. **Important**: Modify the `USERNAME` and `PASSWORD` variables within `server.py` to secure your access credentials.

7. **Start Server**:
   `bash`
   `python server.py`

8. **Verify Deployment**:

9. Access `http://your-server-domain.com:5000` (or your configured port) in a web browser.

10. You should be presented with a login page.

Client Configuration (keylogger.py)

Keylogger Setup

1. **Update Server URL**:
   Modify the `UPLOAD_URL` variable in `keylogger.py` to point to your deployed remote server:
   `python`
   `UPLOAD_URL = "http://your-server-domain.com:5000/upload" # Replace with your actual server domain/IP`
   Note: The `SERVER_SHUTDOWN_URL` is no longer used by the keylogger for automatic shutdown, as server shutdown is now manual via the web interface.

2. **Install Dependencies**:
   `bash`
   `pip install pynput cryptography pillow requests`

3. **Deploy and Run Keylogger**:
   `bash`
   `python keylogger.py`
   The keylogger will automatically check for and start the `server.py` if it's running locally. For remote server deployment, this auto-start mechanism will not apply, and the keylogger will simply attempt to connect to the specified `UPLOAD_URL`.

Web Control Panel Features

Accessing the Control Panel

Open your web browser and navigate to the URL of your deployed server (e.g., `http://your-server-domain.com:5000`).

Login

- You will be prompted to log in using the `USERNAME` and `PASSWORD` configured in your `server.py` file.

Data Management

- **List Sessions**: After successful login, you will see a list of all collected sessions, organized by timestamp.

- **View Session**: Click on a session ID to view the files (logs, keys, screenshots) uploaded during that specific session.
- **Download Files**: Download individual files from a session.
- **Download All**: Download all collected data across all sessions as a single ZIP archive.

Server Shutdown (Manual)

- The web interface provides a mechanism to manually shut down the server. This is typically done by navigating to a specific URL or clicking a button within the interface (depending on the server's implementation).
- **Important**: This action will terminate the server process and all in-memory data will be lost. Ensure you have downloaded all necessary data before initiating a shutdown.

API Endpoints

Data Collection Endpoints

POST /upload

- **Purpose**: Receives keylogger data (encrypted logs, encryption key, screenshots).
- **Method**: `POST` with `multipart/form-data`.
- **Request Body**: Contains `log` (encrypted text), `key` (encryption key), and `screenshot_*` (image files).
- **Response**: JSON indicating upload status and success message.

GET /status

- **Purpose**: Checks if the server is running and operational.
- **Method**: `GET`.
- **Response**: JSON with server status information.

Data Access Endpoints (Authenticated)

GET /

- **Purpose**: Serves the login page for the web control panel.
- **Method**: `GET`.

POST /

- **Purpose**: Handles login attempts for the web control panel.
- **Method**: `POST`.
- **Request Body**: `username` and `password`.
- **Response**: JSON indicating login success or failure.

GET /list

- **Purpose**: Lists all available uploaded sessions.
- **Method**: `GET`.
- **Authentication**: Requires `Authorization: Bearer <PASSWORD>` header.
- **Response**: JSON array of session IDs.

GET /view/

- **Purpose**: Lists files within a specific session.
- **Method**: `GET`.
- **Authentication**: Requires `Authorization: Bearer <PASSWORD>` header.
- **Response**: JSON array of filenames for the given `session_id`.

GET /download//

- **Purpose**: Downloads a specific file from a session.
- **Method**: `GET`.
- **Authentication**: Requires `Authorization: Bearer <PASSWORD>` header.
- **Response**: The raw file content as a download.

GET /download_all

- **Purpose**: Downloads all collected data across all sessions as a ZIP archive.
- **Method**: `GET`.
- **Authentication**: Requires `Authorization: Bearer <PASSWORD>` header.
- **Response**: A ZIP file containing all uploaded data, organized by session.

Configuration Options

Server Configuration (`server.py`)

```
 # server.py settings
app.secret_key = 'your-secure-key' # IMPORTANT: Change this to a strong, unique key
USERNAME = 'admin'              # Default username for web interface
PASSWORD = 'password123'         # Default password for web interface
```

```
# Host and Port are configured in app.run()
# app.run(host='0.0.0.0', port=5000, debug=True)
```

Client Configuration (`keylogger.py`)

```
# keylogger.py settings
UPLOAD_URL = "http://127.0.0.1:5000/upload" # Change to your remote server URL
MOUSE_MOVE_INTERVAL = 1                     # Mouse movement logging interval
```

Data Storage Structure

In-Memory Organization

All collected data (logs, keys, screenshots) are stored in Python dictionaries within the running `server.py` process. Data is organized by session ID (timestamp).

```
# Example of in-memory structure
uploads = {
    "2024-01-01_10-30-00AM": {
        "encrypted_keylog.txt": b"encrypted_log_data",
        "secret.key": b"encryption_key_data",
        "screenshot_20240101_103005_123.png": b"screenshot_image_data",
        "decryptor.py": b"decryptor_script_content"
    },
    "2024-01-01_11-00-00AM": {
        # ... more session data
    }
}
```

Session Management

- **Timestamp-based**: Each upload creates a new entry in the `uploads` dictionary, using a timestamp as the session ID.
- **Ephemeral Data**: Data is **not written to disk** by the server. It resides solely in the server's RAM. This means that if the server process is stopped, restarted, or crashes, all collected data will be permanently lost.
- **No File Conflicts**: In-memory storage inherently avoids file naming conflicts on disk.

Security Considerations

Production Security Measures

1. **Strong Authentication**: **Immediately change the default `USERNAME` and `PASSWORD` in `server.py` to strong, unique credentials.** For production, consider more robust authentication mechanisms (e.g., OAuth, JWT).
2. **HTTPS**: **Crucial for production deployments.** Use SSL/TLS encryption for all communications between the keylogger and the server, and for accessing the web control panel. This protects sensitive data (logs, keys, screenshots, and login credentials) from eavesdropping.
3. **Access Control**: Restrict access to the server and its web interface to authorized personnel only. This can be done via network firewalls, VPNs, or IP whitelisting.
4. **Firewall**: Configure network firewalls to only allow necessary incoming connections to the server's port (e.g., 5000).
5. **Monitoring and Logging**: Implement robust server-side logging and monitoring to detect suspicious activities, unauthorized access attempts, or server issues.

Recommended Security Enhancements

- **Implement a proper user management system** for the web control panel instead of hardcoded credentials.
- **Add rate limiting** to the login endpoint to prevent brute-force attacks.
- **Consider using a dedicated web server** (e.g., Nginx, Apache) as a reverse proxy in front of Flask for better performance and security features.
- **Regularly review and update** all components (keylogger, server, dependencies) to patch vulnerabilities.

Network Security

- **VPN Access**: Consider placing the server behind a VPN, allowing access to the control panel only through the VPN.
- **IP Whitelisting**: Configure your server or firewall to only accept connections from specific, trusted IP addresses.
- **Input Validation**: While Flask handles some, ensure all incoming data is rigorously validated to prevent injection attacks.

Troubleshooting

Common Issues

Server Deployment Issues

1. **Port Conflicts**: If port 5000 is already in use, change the port in `server.py` (`app.run(port=YOUR_PORT)`).
2. **Dependency Issues**: Ensure all required Python packages (`flask` , `cryptography`) are installed on the server.
3. **Authentication Failure**: Double-check the `USERNAME` and `PASSWORD` in `server.py` and ensure they match what you are entering in the web interface.

Client Connection Issues

1. **Network Connectivity**: Verify that the client machine can reach the remote server's IP address/domain and port.
2. **URL Configuration**: Ensure the `UPLOAD_URL` in `keylogger.py` is correctly set to your remote server's address.
3. **Firewall Blocking**: Check firewall rules on both the client and server machines that might be blocking the connection.

Web Interface Issues

1. **Login Problems**: Verify credentials. If you changed them in `server.py` , ensure you're using the new ones.
2. **JavaScript Errors**: Check your browser's developer console for any JavaScript errors that might prevent the interface from functioning correctly.

Debug Mode

For troubleshooting, you can temporarily enable debug mode in `server.py` :

```
 # In server.py
app.run(host='0.0.0.0', port=5000, debug=True)
```

WARNING: Do not run in debug mode in a production environment as it can expose sensitive information and allow remote code execution.

Log Analysis

Monitor server logs (if your webhosting provides them) for:
- Upload success/failure rates.
- Connection attempts.
- Authentication attempts (successful and failed).
- Error patterns.

Best Practices

Deployment Best Practices

1. **Environment Variables**: Use environment variables for sensitive configuration (e.g., `USERNAME`, `PASSWORD`, `SECRET_KEY`) instead of hardcoding them directly in `server.py`.
2. **Process Management**: For production, use a robust process manager (e.g., Gunicorn, uWSGI, Systemd, Docker) to run `server.py` and ensure it restarts automatically if it crashes.
3. **Backup Strategy**: Since data is in-memory, implement a client-side backup or a more robust server-side persistence mechanism if data loss on restart is unacceptable.

Operational Best Practices

1. **Regular Updates**: Keep all software components (Python, Flask, dependencies) updated to the latest versions to benefit from security patches and bug fixes.
2. **Capacity Planning**: Monitor server resource usage (memory, CPU) to ensure it can handle the volume of incoming data, especially since data is stored in-memory.
3. **Data Retention**: Define and enforce a clear data retention policy for collected data.

Development Best Practices

1. **Version Control**: Use Git or similar for managing code changes.
2. **Testing**: Thoroughly test all functionalities, especially data collection and web interface interactions.
3. **Documentation**: Maintain clear and up-to-date documentation.

Legal and Ethical Considerations

Important Disclaimers

1. **Legal Compliance**: Ensure compliance with all local, national, and international laws and regulations regarding data collection, privacy, and surveillance.
2. **Consent Requirements**: Always obtain explicit and informed consent from individuals whose activities are being monitored. This is a legal and ethical imperative.
3. **Data Protection**: Implement appropriate technical and organizational measures to protect collected data from unauthorized access, disclosure, alteration, or destruction.
4. **Responsible Use**: This project is for educational and authorized purposes only. Any unauthorized or malicious use is strictly prohibited and may lead to severe legal consequences.

Recommended Practices

1. **Clear Policies**: Establish clear policies regarding the purpose, scope, and duration of monitoring.
2. **Regular Audits**: Conduct regular security and compliance audits of your system.
3. **Training**: Provide proper training to anyone involved in deploying or managing this system.
4. **Incident Response**: Have a well-defined incident response plan in case of data breaches or security incidents.