Jimsay Private Chat

James Kraemer
Jacob Collins

Contents

1 Introduction

This specification describes a highly specialized protocol for communication between two individuals, James Kraemer, and his wife Lindsay Kraemer. The system consists of a central server and two types of clients.

The first type of client is the web client. The web client opens the server's hosted website via a web browser. The second type of client connects to the server via a TCP connection.

The web client can send simple text messages, or files with a specified recipient to the server. The server then routes the payload to the client.

2 Basic Information

There are two ways to communicate to the server. The first is via HTTP. The server hosts a website that can be accessed via a web browser. The second communication interface is via a TCP/IP connection. The server listens for client connection on port 27272. Only TCP clients in the server's white-list are granted a connection, and a socket stream is opened.

3 Message Processing

There are a few requirements for this protocol. Since the packets are sent as JSON objects, the length can vary, and the length is not included in the details. Instead, packets must include a 16 bit CRC, and be byte stuffed by the sender before going over the wire, and incoming packets must be decoded.

3.1 CRC

3.1.1 Summary

A 16-bit CCITT CRC must be appended to the end of the packet prior to byte stuffing.

3.2 Encoding

3.2.1 Summary

The encoding algorithm must be performed on each packet being sent. This algorithm must run after the CRC is appended to the packet. This process is to ensure that each packet is captured between the necessary encapsulating bytes (0x7E). The following algorithm describes message encoding that must be performed by the sender.

3.2.2 Algorithm

```
frame_byte = 0x7E
escape_byte = 0x7D
xor_byte = 0x20
data = []
data.append(frame_byte)
for each byte in packet do
    if x then
        data.append(escape_byte)
        data.append(byte XOR xor_byte)
    else
        data.append(byte)
    end
end
data.append(frame_byte)
return data
```

3.3 Decoding

3.3.1 Summary

The decoding algorithm can be done each time the socket receives any amount of packets. A data structure containing 1 or more decoded packets is the intended output. The following algorithm describes message decoding that must be performed by the receiver.

3.3.2 Algorithm

```
frame_byte = 0x7E
escape_byte = 0x7D
xor_byte = 0x20
dataArray = []
data = []
i = 0
while i ¡ len(packet) do
    byte = packet[i]
    if byte == frame_byte AND NOT data.empty() then
        dataArray.append(data)
        data = []
    else if byte == escape_byte then
        i += 1 byte = packet[i]
        data.append(byte XOR xor_byte)
    else
        data.append(byte)
    i += 1
end
return dataArray
```

4 Message Infrastructure

4.1 Generic Message Format

{ 'opcode': $< opcode >$, 'payload': $< payload >$ }

4.1.1 Operation Codes

0: JPC_HELLO
1: JPC_CLOSE
2: JPC_HEARTBEAT
3: JPC_ERROR
4: JPC_SEND
5: JPC_TELL

5 Messages

5.1 Hello

{ 'opcode': 0, 'payload': $< MACAddress >$ }

5.1.1 Usage

The JPC_HELLO packet is sent from a TCP client to the server. The TCP client includes their MAC address as an integer for the payload. The server then checks this MAC address against it's white-list, and accepts the connection accordingly.

5.2 Close

{ 'opcode': 1, 'payload': NULL }

5.2.1 Usage

The JPC_CLOSE packet can be sent from server to TCP client, or TCP client to server to indicate that the connection will be closed. The sender and receiver are then expected to close their connections.

5.3 Heartbeat

{ 'opcode': 2, 'payload': $< MACAddress >$ }

5.3.1 Usage

The JPC_HEARTBEAT packet must be sent from server to TCP and vice versa. This packet must be sent atleast once every five seconds. If a five second period elapses without a connection heartbeat, the connection must be closed.

5.4 Send

{ 'opcode': 4, 'payload': < *message* > }

5.4.1 Usage

The JPC_SEND packet is received via the server's hosted site when the HTTP client presses send.

5.4.2 Message

{ 'recipient': < *recipient name* >, 'type': < *message type* >, 'message': < *message* > }

5.4.3 Message Types

0: Plain Text
1: Image

5.5 Tell

{ 'opcode': 5, 'payload': < *message* > }

5.5.1 Usage

The JPC_TELL packet is sent from the server to the appropriate TCP client.

5.5.2 Message

{ 'type': < *message type* >, 'message': < *message* > }

5.5.3 Message Types

0: Plain Text
1: Image

6 Error Handling

6.0.1 Server

The server must be able to handle client crashes. The client socket must be properly disconnected, so that it can reconnect in the event that it is turned back on.

6.0.2 Client

The client must be able to handle server crashes. The server socket must be properly disconnected, so that it can reconnect in the event that the server is turned back on.

7 Asynchronous I/O

The server must be able to handle all clients on the white-list, but not all may be connected at the same time. In order to do so, the server must have a thread (likely the main thread) that accepts new connections, and a new thread to process each client's TCP stream.

8 Conclusion

This specification provides a generic message passing framework for 2 clients to communicate with each other via a central forwarding server.

Without any modifications to this specification, it is possible for clients to devise their own protocols that rely on the text-passing system described here. For example, transfer of arbitrary binary data can be achieved through transcoding to base64. Such infrastructure could be used to transfer arbitrarily large files, or to establish secure connections using cryptographic transport protocols such as Transport Layer Security (TLS).

9 Security Considerations

Messages sent using this system have no protection against inspection, tampering or outright forgery. The server sees all messages that are sent through the use of this service. 'Private' messaging may be easily intercepted by a 3rd party that is able to capture network traffic. Users wishing to use this system for secure communication should use/implement their own user-to-user encryption protocol.