Second question :

1.

|  |  |
| --- | --- |
| Threat | Weak input validation on client username fields |
| Affected component | Client registration and message sending input (handle\_request function) |
| Module details | client.cpp (lines around the handle\_request and create\_registration\_packet functions) |
| Vulnerability class | Input validation bypass / injection potential |
| Description | The program accepts user input for usernames and recipient names without sanitization or length checking. The username is sent directly to the server as part of the binary protocol. Long or invalid characters (such as non-printable characters or oversized strings) can cause protocol corruption or overflow issues on the server side. An attacker could also send crafted binary input through this mechanism. |
| Result | May allow denial of service, corrupted server state, or unexpected behavior, potentially leading to server crashes or vulnerability chaining. |
| Prerequisites | The client allows free text input for usernames or recipient fields without checks. The attacker has access to the client input field. |
| Business impact | Could lead to server instability, crashes, or protocol parsing errors. This could disrupt communication services or be exploited as part of a larger attack. |
| Proposed remediation | Validate input length on the client side before sending to the server (e.g., limit usernames to 16 ASCII characters). Block non-ASCII characters. Do not allow empty strings or overly long usernames. On the server side, reject malformed input. |
| Risk |  |

2.

|  |  |
| --- | --- |
| Threat | Local information leakage through unprotected my.info file |
| Affected component | Local file handling in the client (handle\_response when saving user data) |
| Module details | client.cpp (lines around writing my.info) |
| Vulnerability class | Local file exposure / unencrypted sensitive data |
| Description | The client writes the username and client ID in plain text to a local file called my.info, without encryption or any file protection. This file is stored in a predictable location and can be read by any user or malware on the same system. It contains sensitive identifiers that could be used to impersonate the user or launch further attacks. |
| Result | Any local attacker or malicious process on the same machine can read my.info and obtain client IDs and usernames for impersonation or attacks on the server. |
| Prerequisites | The attacker has access to the client’s filesystem (even without elevated permissions). |
| Business impact | User impersonation, unauthorized actions on the server, and potential compromise of user accounts. |
| Proposed remediation | Store my.info in a protected location (e.g., OS secure storage or user-specific directory with restricted permissions). Consider encrypting client ID data or using obfuscation. |
| Risk | <ul><li>Damage potential: 6</li><li>Reproducibility: 10</li><li>Exploitability: 7</li><li>Affected users: 7</li><li>Discoverability: 9</li><li>Overall: 7.8</li></ul> |

3.

|  |  |
| --- | --- |
| Threat | No verification of server identity (no server authentication) |
| Affected component | Client connection logic in connect\_to\_server() |
| Module details | network.cpp (function connect\_to\_server()) |
| Vulnerability class | Lack of server authentication / Man-in-the-middle risk |
| Description | The client connects to any IP and port provided in server.info without verifying the server’s identity. There is no use of TLS, no certificate checks, and no verification that the server is trusted. An attacker on the same network or who hijacks DNS could impersonate the server and receive client registration details, messages, or other sensitive information. |
| Result | The client may send sensitive information (client ID, usernames, messages) to an attacker instead of the legitimate server. |
| Prerequisites | The attacker needs to perform DNS spoofing, ARP spoofing, or control the network route between client and server. |
| Business impact | Identity theft, user data compromise, message interception, and unauthorized control over client-server communication. |
| Proposed remediation | Add SSL/TLS support, verify server certificates, and only connect to trusted servers. Alternatively, use public key pinning or shared secret verification |
| Risk | <ul><li>Damage potential: 8</li><li>Reproducibility: 9</li><li>Exploitability: 7</li><li>Affected users: 7</li><li>Discoverability: 8</li><li>Overall: 7.8</li></ul> |

4.

|  |  |
| --- | --- |
| Threat | No authentication on message fetching (request 604) |
| Affected component | handle\_client() and process\_request() functions for request code 604 |
| Module details | message\_handler.py (request 604 handling) |
| Vulnerability class | Missing authentication check |
| Description | When a client sends request code 604 (get waiting messages), the server only decodes the client ID from the header and retrieves messages. It does not verify if the client ID belongs to an authenticated or registered user. Any client could send a forged client ID to receive messages that don’t belong to them. |
| Result | Unauthenticated access to private messages for other users. |
| Prerequisites | Attacker knows or can guess another user's client ID (UUID). |
| Business impact | Confidential message leakage, severe privacy violation, and potential impersonation. |
| Proposed remediation | Before serving request 604, check if the client ID exists in user\_storage using user\_storage.get\_user\_by\_id(). Only serve messages to verified clients. |
| Risk | <ul><li>Damage potential: 8</li><li>Reproducibility: 9</li><li>Exploitability: 7</li><li>Affected users: 7</li><li>Discoverability: 8</li><li>Overall: 7.8</li></ul> |
|  |  |

5.

|  |  |
| --- | --- |
| Threat | Fixed-size recv(1024) leads to potential data truncation or partial reads |
| Affected component | handle\_client() function |
| Module details | message\_handler.py (line where data = conn.recv(1024)) |
| Vulnerability class | Incomplete or partial packet processing |
| Description | The server reads incoming client data with a single conn.recv(1024) call. TCP does not guarantee that the entire request will arrive in one chunk or be smaller than 1024 bytes. If the client sends a packet larger than 1024 bytes (or if TCP splits it), the server will only process part of the data, causing protocol errors or corrupted payload parsing. |
| Result | Incorrect or incomplete request handling, protocol decoding errors, and possibly server crashes or unintended behavior. |
| Prerequisites | The client sends a large payload (like a long username or large message) that exceeds 1024 bytes or arrives in multiple TCP packets. |
| Business impact | Potential denial of service or unreliable handling of legitimate client requests. |
| Proposed remediation | Use a loop to receive data until the full payload is read, based on the length from the decoded header, or use socket.makefile() for safe stream reading. |
| Risk | <ul><li>Damage potential: 5</li><li>Reproducibility: 9</li><li>Exploitability: 6</li><li>Affected users: 6</li><li>Discoverability: 9</li><li>Overall: 7.0</li></ul> |