Vulnerability Assessment for File Transfer System

|  |  |
| --- | --- |
| Threat | Information leaked by eavesdropping on the conversation between the client and server. |
| Affected component | Communication |
| Module details | Files: Client.cpp, handle\_client.py. |
| Vulnerability class | Man in the middle attack |
| Description | Except for the file itself, every other piece of information in the payload is not hashed or encrypted. |
| Result | Malicious user eavesdropping on the conversation between the client and the server can learn things like the client ID, name, file name, checksum etc. |
| Prerequisites | Network connection, option to eavesdrop on the conversation |
| Business impact |  |
| Proposed remediation | Start by sharing a private key between the client and server, encrypt all communication payloads from then onwards so the only thing a malicious user can learn is the public RSA key which is useless to them (unless they happen to have a quantum computer). |
| Risk | Damage potential: 5  Reproducibility: 6  Exploitability: 7  Affected users: 9  Disoverability: 8  Overall: 7 |

|  |  |
| --- | --- |
| Threat | Impersonation of other clients |
| Affected component | Client-side login function |
| Module details | Files: Client.cpp, handle\_client.py. Functions: Client::login(), handle\_login() |
| Vulnerability class | Missing credentials, missing digital certificate/signature |
| Description | The only credentials used to login a user are the name and client ID, both of which are relatively easy to get to. |
| Result | A malicious client can find out another client’s name and ID (using the previously explained MITM attack), login as that client, and overwrite their files with their own files (the file name is also accessible from the MITM attack). |
| Prerequisites | Network connection, knowledge of name and client ID |
| Business impact |  |
| Proposed remediation | Add a hashed digital signature/certificate or a hashed password, any authentication token that cannot be picked up just from eavesdropping. |
| Risk | Damage potential: 6  Reproducibility: 8  Exploitability: 9  Affected users: 7  Disoverability: 9  Overall: 7.8 |

|  |  |
| --- | --- |
| Threat | Malicious user learning private key from me.info or priv.key |
| Affected component | Client information saving |
| Module details | File: Client.cpp. function: Client::saveClientInfo(), Client::savePrivateKey() |
| Vulnerability class | Sensitive information saved locally |
| Description | The client stores its private RSA key locally and in a cryptographically unsecure fashion (base64 is not cryptographically secure) in me.info and priv.key |
| Result | If an attacker ever gains the ability to read specific files, or socially engineers the user into revealing them to him, they can learn the private RSA key which is used in all subsequent communication between the client and the server – thus being able to decode the files and save them for himself (maybe using the previously explained MITM attack). |
| Prerequisites | Access to local files of the client |
| Business impact |  |
| Proposed remediation | Don’t save the key locally – create a new key each time instead. If you really want to use the same key, have the server generate and save it instead of the client, assuming the defenses for the server are more robust. |
| Risk | Damage potential: 10  Reproducibility: 6  Exploitability: 7  Affected users: 6  Disoverability: 8  Overall: 7.4 |

|  |  |
| --- | --- |
| Threat | Denial of service because of client overload |
| Affected component | Server that handles client requests |
| Module details | Files: server.py. Function: start\_server() |
| Vulnerability class | Denial of service vulnerability |
| Description | Every single client receives their own thread. |
| Result | If a malicious attacker wants to cause a DoS, they can create a huge amount of clients to overload and crash the server. Limiting the amount of clients can also cause denial of service since a malicious attacker can fill the quota with their own clients. |
| Prerequisites | Network connection |
| Business impact |  |
| Proposed remediation | Limit the amount of concurrent users anyways, and try to have them share threads and support switching between them.  Another option is splitting the server into multiple microservices, each of which deals with the client thread separately so you can’t overload the entire server.  Finally, limit the number of connections from specific IP addresses. This will cause the attacker to need many virtual machines and VPNs, making it much more difficult to exploit the vulnerability. |
| Risk | Damage potential: 5  Reproducibility: 8  Exploitability: 6  Affected users: 9  Disoverability: 8  Overall: 7.2 |