Security use-cases

1. Útok na spojenie medzi klientom a serverom alebo medzi 2 klientmi.

1.1. Strata správy pri prenose po sieti.
Problém: Správa sa stratí prirodzeným procesom ako dôsledok nedokonalosti prenosového média alebo chyby v aktívnych prvkoch (reuter, switch, AP, ...)
Ochrana:
Použitie protokolu TCP, ktorý zaručuje dokonalú službu prenosu dát na úkor rýchlosti/efektívnosti.

1.2. Zablokovanie správy útočníkom.
Problém: Útočník zachytí správu a nepošle ju ďalej.
Ochrana:
Ochrana šifrovaním správy aby útočník nemohol pochopiť jej obsah. Momentálne prenos pomocou SSL TCP socketu.
SSL socket by mal použiť TLS 1.2 protokol a pre šifrovanie používať najlepšie RSA pre asymetrickú a AES pre symetrickú kryptografiu. SSL 3 je prekonané a nie je bezpečné.

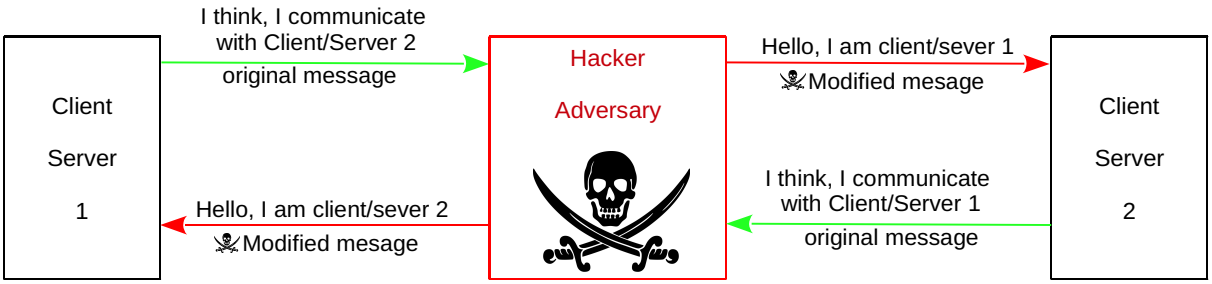
https://cs.wikipedia.org/wiki/Transport_Layer_Security

http://doc.qt.io/qt-5/qsslsocket.html

1.3. Man in the middle útok
Problém: Útočník odchytáva komunikáciu na zdieľanom prenosovom médiu.

1.3.1. Útočník odpočúva komunikáciu
Ochrana: rovnako ako v prípade 1.2.

1.3.2. Útočník odpočúva komunikáciu, snaží sa ju zmeniť a pre obidve strany sa snaží predstierať, že je skutočný užívateľ.



Problém: Útočník odchytáva komunikáciu od stanice A, snaží sa pre ňu tváriť, že je stanica B a stanici B posiela buď pozmenené správy tak aby si myslela, že útočník je stanica A alebo stanica B vôbec nevie o prebiehajúcej komunikácii.
Ochrana:
Použitie vhodného protokolu z rodiny Challenge-handshake authentication protocol. Obdive stanice najprv nadviažu spojenie pomocou asymetrickej kryptografie, následne si týmto šifrovaným spojením prepošlú zdieľaný kľúč. Pomocou zdieľaného kľúča vytvoria spojenie šifrované symetrickou šifrou. Pre prípad prelomenia symetrickej šifry obsahujú správy digitálny podpis ako nástroj detekcie porušenia

integrity správy.
Digitálny podpis je hash kontrolného súčtu posielanej správy zašifrovaný súkromným kľúčom odosielateľa.

Poznámka:
Protokoly MS CHAP, ktoré patria do rodiny Challenge-Handshake Authentication Protocols sú nevhodné kvôli zraniteľnosti použitého DES šifrovania. Vhodnejšie by bolo asi použiť protokol OpenID alebo nejaký iný.

https://en.wikipedia.org/wiki/Authentication

https://en.wikipedia.org/wiki/MS-CHAP

1.3.3. Útočník pošle starú zachytenú správu
Ochrana: Všetky správy sú opatrené timestampom a obidve stanice si vedú zoznam aktívnych spojení. V prípade, že správa nepatrí k žiadnemu aktívnemu spojeniu alebo je staršia ako posledná prijatá správa rámci aktívneho spojenia, tak je považovaná za duplicitnú a ignorovaná/zahodená.

1.3.4. Útočník sa pokúša nadviazať spojenie s jednou zo staníc v sieti a vydáva sa za niektorú z ostatných staníc.
Ochrana: Rovnako ako v prípade 1.3.2. Stanice nadväzujú spojenie pomocou Challenge-response protokolu. Rámci tohoto protokolu sa využíva asymetrickej kryptografie, kde každá zo staníc dokáže byť identifikovateľná pomocou znalosti správneho hesla a následne už automaticky pomocou dokázania znalosti svojho súkromného kľúča. Zároveň pre tieto účely by mal server obsahovať register verejných kľúčov jednotlivých klientov a všetci klienti by mali mať verejný kľúč serveru.

Poznámka:
Po každej úspešnej registrácii, resp. prihlásení sa si klient vygeneruje vlastnú dvojicu verejný, súkromný kľúč a so serverom si navzájom aktualizujú verejné kľúče.

1.3.5. Útočník pozmení správu bez znalosti jej obsahu/sémantiky
Ochrana: Kontrolný súčet z digitálneho podpisu zaručuje detekciu narušenia integrity správy.

1.3.6. DoS útok
Problém: Útočník zaplaví jednu zo staníc takým množstvom správ, že nie je schopná prijať korektnú komunikáciu.
Ochrana: To by bolo na samostatný dokument. Momentálne by som to riešil vytvorením whitelistu na ktorom budú IP adresy servera a užívateľov z friendlistu. Všetky ostatné stanice budú automaticky blokované.

2. Útok na klienta

2.1. Snaha získať heslo užívateľa klienta
Ochrana: V prvom rade si musí užívateľ zabezpečiť vlastnú počítačovú stanicu voči útokom typu: keylogger, vírus, phishing, apod. V druhom rade klient nikdy nebude ukladať užívateľove heslo. Autentizácia sa bude konať s pomocou servera, ktorý bude držať databázu hesiel užívateľov v zahashovanej, osolenej a prípadne aj zašifrovanej forme. Teoreticky môže klient tiež podobne uložiť heslo, aby mohol užívateľ zapnúť aplikáciu offline, otázkou je či je to požadovaná vlastnosť.

2.2. Snaha vydávať sa klienta ktorým útočník nieje.
Ochrana: Užívateľ sa pri komunikácii autentizuje heslom prípadne iným spôsobom.

2.3. Úplná nedostupnosť klienta
Problém: útočník vyradí z činnosti klienta
Ochrana: z pohľadu developera neexistuje ochrana, o toto sa musí postarať užívateľ daného klienta. Z hľadiska fungovania celej siete toto nieje nebezpečná udalosť, lebo je ekvivalentná so stavom, kde sa klient sám odpojí.

3. Útok na server

3.1. Pokus o prevzatie kontroly na serverom
Ochrana: štandartné ochrany pre serveri. Server by si mal ukladať ACL listy administrátorov, ktorý majú právo jeho úprav.

3.2. Pokus o ukradnutie databázy hesiel
Ochrana: Databáza neobsahuje heslá v otvorenej forme ale obsahuje iba hashe osolených hesiel, zašifrované pomocou verejného kľúča serveru.

3.3. Úplná nedostupnosť serveru
Problém: útočník vyradí nejakým spôsobom server z činnosti
Ochrana:
a) Ak sa 2 klienti chcú spojiť a majú o sebe záznamy vo vlastných friendlistoch a friendlisty sú aktuálne (neobsahujú zastaralú IP adresu). Tak by nemal byť so spojením problém. Ale väčšinou bude kvôli dynamicky pridelovaným adresám v súčasnom nedokonalom IPv4 svete.
b) Teoreticky by bolo možné vytvoriť procedúry, tak aby každý klient po prihlásení do siete obdržal od serveru databázu hesiel, zoznam užívateľov a prevzal rolu serveru akonáhle by sa mu po určitom timeoute neozval. Avšak otázka je či by takéto zvýšenie robustnosti siete, posunutím do roviny čistej P2P siete neznížilo úroveň zabezpečenia jednotlivých klientov, vzhľadom na to, že by zo serverom zdieľali databázu hesiel.
c) Zvýšiť počet serverov, čím sa zvýši robustnosť siete a aj keby sa útočníkovi podarilo zhodiť všetky servery, tak čo už, nahodíme a sieť nebude schopná určitý čas vytvárať nové spojenia medzi klientmi, avšak nadviazané spojenia to nenaruší.

Poznámka:
Ak by pre vytvorenie serveru stačilo zdielať iba zoznam užívateľov a aktívnych spojení, tak by sa klienti mohli stať servermi bez toho aby server musel prezradiť databázu hesiel.

V takomto prípade by sa mohlo heslo ukladať aj lokálne v klientovi. Klient by sa potreboval spojiť so serverom iba v nasledujúcich prípadoch:
a) registrácia nového užívateľa
b) prvé nadviazanie spojenia s neznámym klientom (klient by si mohol ukladať staré spojenia)
c) administratívne požiadavky ako napr. zmena hesla.

Je isté, že vyradenie serveru je najväčšia zraniteľnosť našej siete.

Dôležité poznámky pre API:

1. Žiadny klient si neukladá svoj pár verejný/súkromný kľúč. Vždy si ich generuje pri logovaní a po úspešnej autentizácii updatuje svoj verejný kľúč v databáze serveru.

2. Treba vytvoriť login serveru pre administrátorov, tak ako to má aj klientská aplikácia.

3. Pozor na protokol použitý SSL socketom.

4. Prvý výskyt stavovosti. Je možné, že server si bude musieť udržiavať zoznam aktívnych spojení , ale niesom si istý, či túto funkcionalitu nepokryje ten SSL TCP socket v QTčku.

TODO:

1. Upresniť verziu AES a RSA pre SSL Socket

2. Zistiť ktorú časť bezpečnosti treba implementovať a ktorú časť už sám má v sebe implementovaný Qt knižnica, ako napr. tomu je u SSL socketu.

3. Rozhodnúť ako veľmi robustnú sieť chceme voči výpadku servera.