1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

UNIVERSITÀ DEGLI STUDI DI URBINO CARLO BO
DIPARTIMENTO DI SCIENZE PURE E APPLICATE
CORSO DI LAUREA DI INFORMATICA APPLICATA

Progetto d'esame di reti di calcolatori

Marzio Della Bosca

Matricola: 306034

Indice

1

Specifica di progetto

Lo scopo del progetto è quello di implementare un architettura client-server in cui un entità server mette in comunicazione un numero indefinito di client. Le entità che comunicano per mezzo del server sono di due tipologie:

- Client: Semplice client in grado di scambiare messaggi con gli altri client connessi al server. L'entità Client 'modella' un dipendente che utilizza il sistema.
- Admin: Un client in grado di eseguire delle operazioni speciali come chiudere la comunicazione del client al server, eseguire comandi con permessi elevati sul dispositivo di un client connesso ed effettuare una scansione delle porte TCP dei client. L'entità Admin 'modella'

Questo sistema è stato pensato come una possibile applicazione su di una rete aziendale infatti il sistema non è in grado di uscire su internet. Lo scopo del progetto è quello di consentire ai dipendenti connessi al sistema di ricevere un primo aiuto da parte di un tecnico in modo che se un dipendende riscontra problemi nell'utilizzo del propio dispositivo può chiedere un intervendo immediato di diagnosi. Le funzionalità messe a disposizione del Admin sono in grado di fare una prima valutazione dei problemi che intercorrono sul dispositivo del richiedente.

2

Analisi del problema

Data la specifica è possibile dividere il problema in 4 aree principali:

- Attori
- Comunicazione tra i client
- Concorrenza
- Sicurezza

Per quanto riguarda il problema relativo al numero di attori, i client che si connettono al server, anche se presentano anlcuni tratti in comune, hanno caratteristiche e necessità diverse, prendendo in considerazione ciò che hanno bisogno di fare i tecnici rispetto agli altri dipendenti, quindi è necessario creare due entità distinte oltre all'entità che servirà da intermediario nella comunicazione tra i client.

La comunicazione tra i client sarà gestita dall'entità del server, che rimarrà in ascolto e al contempo in attesa di recapitare i messaggi ai client. Si rende necessaria poi la gestione della concorrenza, dato che il server parallelamente deve restare in ascolto dei messaggi dei client e recapitarli, mentre da lato client è necessario rimanere in ascolto delle comunicazioni da parte del server e prendere in input da tastiera i messaggi. La divisione della entità Client in due entità è già una prima forma di sicurezza perchè si ha più controllo sulla gestione della rete in quanto solo alcuni dipendenti avranno accesso alle entità admin, per il quale è necessaria l'immissione di una password per il suo utilizzo. Dividere poi mantiene il codice più ordinato e facilmente aggiornabile.

La verifica della password verrà fatta lato server, sempre per questioni relative alla sicurezza, e tutti i comandi 'speciali' sono consentiti solo agli admin, inoltre i client normali non potranno avere il nominativo riservato 'admin'. Dato che la comunicazione intercorre tra client su dispositivi diversi, anche se sulla stessa rete, è necessario lavorare sulle regole di firewall che consentono (o meno) il traffico in entrata/uscita sulla porta specifica del dispositivo su cui opera il server. Un altra misura di sicurezza quindi è quella di chiudere la porta di comunicazione TCP a fine esecuzione quando il server è in chiusura. Infine i messaggi scambiati nel sistema verrano criptati, dato che c'è un passaggio di informazioni sensibili come la password dell'admin, il report della scansione delle porte tcp e l'output dei comandi lanciati dall'admin sul dispositivo del client.

3

Scelte di progetto

Data la natura del progetto ho deciso di implementarlo usando il linguaggio python per via della sua versatilità dato il grande numero di librerie messe a disposizione. Ho utilizzato un approccio Object Oriented in cui, come precedentemente accennato nella sezione di specifica, il modello consiste di tre entità distinte: Server, Client e Admin. L'entità Admin può essere vista come una specializzazione dell'entità Client

La libreria Scapy che è una libreria molto utilizzata nell'ambito della sicurezza informatica e per il networking. Dato che i client (sia admin che client normali) non sono presenti, ovviamente, sulla stessa macchina del server e dato che non conoscono l'indirizzo privato di esso questi devono cercarlo. Ho utilizzato scapy per fare questa operazione perchè si è rivelato molto più veloce rispetto alle altre librerie provate, per quanto riguarda la ricerca dei dispositivi collegati alla rete, e dato che il progetto sarebbe pensato in chiave aziendale sarebbe stata un operazione lunghissima prendendo in considerazione che il processo nella rete domestica (in cui al massimo ci sono collegati contemporaneamente 5/6 dispositivi) su cui sono state fatte le prove richiedeva tempistiche di minuti. Scapy è stato utilizzato anche nella implementazione dello scanner TCP, molto utile in quanto ha permesso di creare dei pacchetti personalizzati, un 'must' in quanto i firewall possono effettuare delle tecniche di filtraggio dei pacchetti TCP utilizzando come discrimine le porte sorgenti, sequenze TCP ecc.

Per la scannerizzazione delle porte TCP da parte dell'admin ho fornito una lista delle porte TCP su cui girano i servizi più utilizzati: Grazie per aver fornito la lista delle porte e dei relativi servizi. Ecco una descrizione di ciascuna porta e del servizio associato:

- Porta 20: FTP data, utilizzata per il trasferimento dei dati da e verso un server FTP.
- Porta 21: FTP, utilizzata per la comunicazione di controllo tra un client FTP e un server FTP.
- Porta 22: SSH, utilizzata per connettersi a un server utilizzando il protocollo SSH.
- Porta 23: Telnet, utilizzata per la connessione a un server Telnet, che consente di accedere a un altro computer da remoto.
- Porta 25: SMTP, utilizzata per l'invio di e-mail attraverso un server SMTP.
- Porta 37: Time, utilizzata per sincronizzare l'orologio di un computer.
- Porta 53: DNS, utilizzata per la comunicazione con un server DNS, che fornisce informazioni sulla risoluzione dei nomi di dominio.
- Porte 67 e 68: DHCP Server, utilizzata per la comunicazione con un server DHCP che assegna dinamicamente gli indirizzi IP ai dispositivi connessi alla rete.
- Porta 69: TFTP, utilizzata per il trasferimento di file (Trivial File Transfer Protocol).
- Porta 80: HTTP, utilizzata per la comunicazione con un server HTTP, fornisce l'accesso ai siti web.
- Porta 88: Kerberos, utilizzata per la comunicazione con un server Kerberos, fornisce un sistema di autenticazione sicuro.

- Porta 110: POP3, utilizzata per la ricezione di e-mail attraverso un server POP3 (Post Office Protocol versione 3).
- Porta 119: NNTP, utilizzata per la comunicazione con un server NNTP, fornisce l'accesso ai newsgroup di Usenet.
- Porta 123: NTP, utilizzata per la sincronizzazione dell'orologio di un computer con un server NTP (Network Time Protocol).
- Porta 137: NetBIOS, utilizzata per la comunicazione tra dispositivi di rete utilizzando il protocollo NetBIOS.
- Porta 143: IMAP, utilizzata per la ricezione di e-mail attraverso un server IMAP (Internet Message Access Protocol).
- Porta 161: SNMP, utilizzata per la comunicazione con un server SNMP, fornisce informazioni sullo stato dei dispositivi di rete.
- Porta 162: SNMP Traps, utilizzata per la comunicazione con un server SNMP, fornisce notifiche di eventi per i dispositivi di rete.
- Porta 179: BGP, utilizzata per la comunicazione tra router, utilizzando il protocollo BGP (Border Gateway Protocol) che consente di scambiare informazioni sulla struttura della rete.
- Porta 389: LDAP, utilizzata per la comunicazione con un server LDAP, fornisce informazioni sulle risorse della rete.
- Porta 443: HTTPS, utilizzata per la comunicazione sicura con un server HTTPS, fornisce l'accesso ai siti web tramite il protocollo HTTPS (HTTP Secure).
- Porta 465: SMTPS, utilizzata per l'invio di e-mail attraverso un server SMTP utilizzando il protocollo SMTPS (SMTP sicuro), fornisce una comunicazione sicura per la trasmissione di e-mail.
- Porta 512: Rexec, utilizzata per la comunicazione con un server Rexec, che consente di eseguire comandi da remoto.
- Porta 513: Rlogin, utilizzata per la comunicazione con un server Rlogin, che consente di accedere da remoto.
- Porta 514: RSH, utilizzata per la comunicazione con un server RSH, che consente di eseguire comandi da remoto.
- Porta 543: KKRPM, utilizzata per la comunicazione con un server KKRPM che fornisce informazioni sulla gestione dei pacchetti software.
- Porta 544: KKRPMP, utilizzata per la comunicazione con un server KKRPMP, fornisce informazioni sulla gestione dei pacchetti software.
- Porta 547: DHCPArchive, utilizzata per la comunicazione con un server DHCP, archivia le informazioni sui client DHCP.
- Porta 548: AFP, utilizzata per la comunicazione con un server AFP, fornisce l'accesso ai file condivisi su una rete Apple.
- Porta 554: RTSP, utilizzata per la comunicazione con un server RTSP, che consente di trasmettere in streaming audio e video su una rete.

- Porta 563: NNTP over SSL, utilizzata per la comunicazione sicura con un server NNTP, fornisce l'accesso ai newsgroup di Usenet tramite il protocollo NNTP (Network News Transfer Protocol) sicuro.
- Porta 587: SMTP Auth, utilizzata per l'invio di e-mail attraverso un server SMTP con autenticazione, che richiede una verifica dell'identità del mittente.
- Porta 636: LDAP over SSL, utilizzata per la comunicazione sicura con un server LDAP, fornisce informazioni sulle risorse di rete tramite il protocollo LDAP (Lightweight Directory Access Protocol) sicuro.
- Porta 993: IMAP over SSL, utilizzata per la ricezione di e-mail attraverso un server IMAP, fornisce un accesso remoto sicuro alle caselle di posta.
- Porta 995: POP3 over SSL, utilizzata per la ricezione di e-mail attraverso un server POP3, che fornisce un accesso remoto sicuro alle caselle di posta.

Come tecnica crittografica ho deciso di utilizzare la crittografia a chiave asimmetrica RSA. Questa utilizza una coppia di chiavi per cifrare e decifrare i messaggi che i client si vanno a scambiare, in cui una chiave è pubblica e l'altra è privata. La chiave pubblica viene condivisa con tutti i client connessi al sistema mentre la chiave privata viene utilizzata solo nella decifratura del messaggio e deve rimanere segreta, quindi non viene condivisa da nessuno. Ho scelto di utilizzare una chiave asimmetrica per questioni di sicurezza dato che nel caso di una crittografia a chiave simmetrica la chiave di cifratura sarebbe la stessa per tutti e quindi in caso di attacco basterebbe accederci per avere sotto controllo tutto il traffico dati del sistema. In questo modo ogni singolo client ha la propia chiave personale, il che rende il sistema più 'robusto' a questo tipo di attacchi. Inoltre nel caso della crittografia simmetrica ci sarebbe l'ulteriore problema della condivisione tra tutti i client, in modo sicuro, della chiave.

Dato che tutte le entità implementate durante la loro esecuzione hanno accesso alla libreria scapy o comunque devono poter eseguire processi che necessitano permessi elevati, sono stati implementati in modo che il loro avvio avviene solamente con privilegi elevati, in modo che poi, durante l'esecuzione, non è più necessario inserire le credenziali amministrative (molto utile nel caso in cui l'admin esegue comandi da terminale su client).

4

Progettazione dell'algoritmo

Il sistema è stato progettato in linguaggio python con un approccio OO (Object Oriented). Le entità che si sono rese necessarie allo sviluppo del progetto sono:

- Server
- Admin
- Client

L'entità server ha lo scopo di mettere in comunicazione i client e autenticare i client di tipo Admin. Quando il server inizia ad eseguire si mette in ascolto su una porta scelta in maniera arbitraria e rimane in ascolto in attesa di client che si collegano. Il server accetta tutte le connessioni e principalmente ha 4 compiti:

1. Fa il binding della socket sulla porta 60000 e si mette in ascolto in attesa di richieste di connessioni, che vengono accettate automaticamente e successivamente c'è lo scambio della chiave pubblica.
2. Quando un client manda un messaggio il server inoltra il messaggio a tutti gli altri client connessi alla rete
3. Quando un client admin tenta di connettersi richiede la password al client e la verifica
4. Gestice i comandi da parte dell'admin

L'entità Client è il software pensato per un dipendente connesso al sistema. Quando il programma parte il client cerca il server all'interno della propia rete locale inviando un messaggio broadcast ARP, poi tenta a connettersi sulla porta specifica 60000, ad ogni host che si è dimostrato raggiungibile, e se il server è attivo ci si collega. L'entità Admin come Client appena

esegue, dopo essersi autenticato, trova il server nella rete locale, e manda e riceve messaggi, però è in grado di mandare comandi speciali al server in modo che admin abbia accesso diretto al client di interesse. Come funzionalità aggiuntiva Admin è in grado di fare una scansione TCP delle porte di un particolare client, in modo che sia possibile verificare quali porte/servizi siano attive su quella particolare macchina. In modo da consentire al Admin di interagire ad un livello più alto il server deve poter riconoscere quali messaggi di Admin siano comandi speciali è necessario implementare un sistema di valutazione del messaggio da parte del server. Tra i comandi speciali disponibili ad Admin troviamo anche le operazioni di 'kick' e 'ban', che anche se possono risultare 'in più' per il contesto di specifica sono importanti a fini di debug, per testare la stabilità del sistema. La codifica scelta nella comunicazione tra i client e il server è la

codifica ASCII. Inoltre è stato scelto di implementare una piccola funzionalità di 'troubleshooting', ovvero uno scan delle porte TCP su un client da parte di un admin al fine di vedere quali sono i servizi attivi sulla macchina del client. Se una porta risponde con un pacchetto di ACK (acknowledgement), significa che la porta è aperta e sta ascoltando le connessioni in ingresso. Ciò suggerisce che un servizio in esecuzione sull'host sta utilizzando quella porta per accettare le connessioni. Quando si effettua una scansione delle porte TCP di un host, si inviano pacchetti TCP alle porte dell'host utilizzate dai servizi più comuni come ad esempio telnet, http, ssh ecc. Per ogni porta la scansione può dare diversi risultati:

- Aperta: la porta TCP è aperta quindi il servizio ad essa associato è in esecuzione e accetta le connessioni.
- Chiusa: la porta TCP è chiusa quindi il servizio non è attivo, non è stata ricevuta risposta.
- Chiusa con messaggio: la porta TCP è aperta però per un qualche motivo non accetta la connessione, per esempio potrebbe essere dato dal fatto che la porta è filtrata dal firewall. Ad esempio il messaggio potrebbe essere 'RA' ovvero 'reset and ack' che significa che la connessione è stata resettata.

Lo scambio delle chiavi pubbliche viene fatto appena il tentativo di connessione al server da parte del client va a buon fine. Inoltre dato che la gestione dei client, da parte del server, viene effettuata sfruttando gli indici delle liste contenenti le informazioni dei client, tutte le informazioni di questi vengono tolte dalle suddette liste quando il client si sconnette. Questo vale sia per le socket, nickname, chiavi pubbliche e indirizzi.

5

Implementazione dell'algoritmo

L'implementazione del'algoritmo è stata divisa in tre parti, una per Server, una per il Client e una per Admin.

Server:
La classe Server possiede 9 metodi, oltre al costruttore della classe, La classe Server prende come parametro la porta su cui si metterà in ascolto il server, alla creazione il server crea una socket ipv4 TCP, farà il binding della socket creata sulla porta e l'host della macchina su cui esegue. Genera la chiave pubblica e la chiave priva ed infine si mette in ascolto aprendo la comunicazione TCP sulla porta e avvia il thread di gestione dei client. I restanti metodi sono:

1. Broadcast: Prende come argomento il messaggio e il client mittente e lo invia a tutti i client connessi meno il mittente.

2. Handler: Gestisce la ricezione messaggi dei client, riconoscendo se il messaggio è un comando speciale da parte di un admin o se è un semplice messaggio. Nel primo caso richiama la funzione 'admin handler' nel secondo la funzione broadcast. NOTA: Per gestire la chiusura corretta del server è stato necessario lavorare sulla gestione delle eccezioni perchè alla terminazione da tastiera del server questo non terminava in attesa del recive del client, quindi il server andava in crash e non terminava correttamente quando un client si disconnetteva dopo che il server era bloccato, è stata usata eccezione socket.timeout per permettere la rivalutazione dell'if ogni 0.5 secondi o while restava bloccato in attesa.

3. Admin Handler: Prende come parametri client e messaggio, il client rappresenta l'admin che ha fatto un comando mentre nel messaggio è presente il nickname target e il tipo di comando richiesto, i comandi permessi sono: kick, ban, su e tcp. I comandi speciali vengono riconosciuti perchè iniziano con il carattere speciale '/'. Il comando 'su' manda un messaggio al client contenente il comando che l'admin ha bisogno di inserire nel terminale del dispositivo del client target e l'output viene mandato dal client al server e poi inoltrato al admin che ne vede la stampa, di modo che admin abbia accesso a una grande mole di informazioni. I comandi ban e kick sono stati utili nella fase di test, in modo da verificare la stabilità del sistema in diversi scenari d'esecuzione. Il comando tcp invece fa si che il server rimandi al admin l'indirizzo associato al nickname richiesto.

4. Receive: Gestisce la connessione dei client, quando un client richiede la connessione questa viene accettata, si richiede il nickname del nuovo client oppure si richiede la password e se ne fa una verifica nel caso il client sia un admin. Infine si fa partire un thread il cui comportamento è descritto dal metodo handler così da avere una relazione client - thread (handler) 1 a 1.

5. Remove client: Questo metodo rimuove i dati del client e chiude la connessione.

6. Get private ip: Ottiene l'indirizzo ip privato della macchina su cui esegue.

7. Port handler: Metodo che gestisce la apertura/chiusura della porta su cui deve rimanere in ascolto.

8. Kick user: Metodo che consente al server la gestione dell'operazione di esclusione dal sistema da parte di un admin nei confronti di un client, non admin. Utilizzo a fini di test.

9. End server: Consente la terminazione controllata del server e infine richiama il metodo port handler per chiudere la porta di comunicazione.

Client:
La classe Client è composta da 3 metodi oltre al costruttore. Il Client alla sua ceazione per prima cosa genera la chiave pubblica e la chiave privata, successivamente crea una socket ipv4 TCP e chiama il metodo 'device finder' per cercare il server, quando lo trova ci si connetta ed effettua lo scambio delle chiavi pubbliche. Successivamente chiede il nickname col quale accedere al sistema, controllando che il nickname di accesso sia diverso da admin (nominativo riservato). Infine crea e fa partire i thread di ricezione messaggi dal server e ricezione input da tastiera i cui comportamenti sono rispettivamente definiti dai metodi receive e write.

1. Receive: Gestisce i messaggi ricevuti dal server in loop, discriminando i messaggi di servizio ricevuti dal server, sopratutto nel caso del comando su che in background il client deve eseguire sulla macchina il comando ricevuto dal server e inviare a quest'ultimo l'output.

2. Write: Gestisce gli input da tastiera.

3. Device finder: Questo metodo cerca il server sulla rete mediante l'utilizzo delle librerie scapy. Invia un pacchetto arp broadcast in modo da vedere gli indirizzi ip della wlan raggiungibili e poi tenta di connettere una socket alla porta su cui è in ascolto il server tentando su ogni host. Per diminuire i tempi di attesa al raggiungimento del server è stato iterato sugli indirizzi ip attuando una reverse sulla lista in quando generalmente sui primi indirizzi della rete ci sarà il router o altri dispositivi precedentemente collegati. L'iterazione sugli host disponibili termina quando non è stato trovato il server, e in tal caso da errore. Oppure quando la connessione alla socket è andata a buon fine.

Admin:
La classe Admin può essere vista come una specializzazione della classe Client. Il costruttore rispetto a Client differisce dal fatto che al posto di richiedere il nominativo, è richiesta la password e viene inviata al server e se la verifica non passa il programma termina. Il metodo 'device finder' resta inalterato mentre i metodi 'write' e 'receive' vengono modificati. Ecco l'elenco dei metodi modificati e aggiunti, non presenti in client:

1. Receive: Come il client riceve i messaggi dal server, le differenze stanno nella tipologia dei messaggi di servizio risolvibili. Qui viene gestito il caso in cui l'admin abbia già fatto una richiesta tcp al server e questo gli stia tornando l'indirizzo IP del client target, in quel caso viene chiamato il metodo 'tcp port scanner'.

2. Write: L'unica differenza con la classe Client sta nel fatto che si esegue un controllo sull'input da tastiera, richiama la funzione 'special commands' nel caso l'admin stia richiedendo al server un comando speciale.

3. Tcp port scanner: Questo metodo utilizza le librerie messe a disposizione da scapy in modo da eseguire una scansione tcp delle porte utilizzate dai servizi più comuni di uno specifico host. Crea un pacchetto TCP per ogni porta (è contenuta una mappa porta/servizio in ogni Admin) con il flag SYN impostato per aprire una connessione. Se la risposta è ricevuta entro il timeout il pacchetto di risposta viene analizzato per determinare se la porta è aperta o chiusa. Se la porta è aperta, viene stampato che il servizio associato alla porta è online. In caso contrario che la porta è chiusa e che il servizio associato alla porta è offline.

4. Special commands: Prende per argomento il messaggio di input. Questo viene analizzato e a seconda del comando riconosciuto invia uno specifico messaggio di servizio al server.

Tutti i messaggi inviati tra Server - Client - Admin sono criptati attraverso la crittografia a chiave asimmetrica RSA e si utilizza la decodifica ascii. Inoltre ho impostato il limite di dimensione dei messaggi a 1024 byte, questa è una buona norma di sicurezza per evitare attacchi di diverso genere come gli attacchi di buffer overflow che si può verificare quando il programma riceve una mole di dati in ingresso tale da saturare e superare il buffer, questo potrebbe consentire ad un attaccante di sovrascrivere parte della memoria del sistema e causare il blocco del programma o l'esecuzione di codice malevolo. Dato che sto utilizzando la codifica ascii a 8 bit, in cui ogni carattere quindi occupa un byte, questo limite consente l'invio di messaggi che arrivano a contenere fino a 1024 caratteri, il che è più che sufficiente per il contesto di specifica.

6

Test del programma

La fase di test è stata condotta in modo da andare a verificare la corretta funzionalità del programma nel corso della sua esecuzione, la strategia utilizzata è stata quella del 'bottom-up', ovvero sono partito ad analizzare il funzionamento dei moduli più semplici fino ad arrivare a quelli più complessi. Infine è stato testato il sistema nel suo insieme al fine di verificare la corretta comunicazione tra i moduli. Sono stati condotti alcuni stress test, in modo da verificare la scalabilità e la stabilità del sistema (molto utili sono stati i comandi speciali di 'kick' e 'ban' che hanno permesso di stabilizzare il sistema generando degli eventi e delle casistiche che hanno sottolineato la necessità di fare della gestione delle eccezioni il principale metodo di controllo del sistema).

Data la natura del sistema, che opera gestendo la concorrenza tra i client, la difficoltà maggiore riscontrata durante l'implementazione del progetto è stata il far comunicare tra loro i moduli facendo attenzione a 'limitare i danni', cioè facendo attenzione al fatto che se uno dei client riscontrava un errore allora solo lui andava in 'crash' e si disconnetteva, senza fare andare in errore il server o gli altri client, ad esempio quando il server si disconnetteva prima dei client questo rimaneva in attesa dei messaggi dei client e andava in crash quando poi anche un client andava a sconnettersi facendo si che il programma non terminava correttamente e le porte di comunicazione non venivano chiuse, quindi si è reso necessario rendere il metodo 'handler' del server non bloccante inserendo un eccezione 'socket.timeout' in modo che il server non si "addormenti" sul comando .recv() in attesa di messaggi.

Il sistema si è rivelato molto scalabile (per i test condotti), ovvero che la sua esecuzione non risentiva del numero di client connessi al server. I test condotti su client e admin hanno verificato la corretta comunicazione col server, come reagivano agli eventi innescati da loro o da altri client/admin e la loro corretta terminazione. Inoltre è stato verificato, mediante l'utilizzo del tool wireshark, che i messaggi scambiati tra i client e il server non venivano mandati in chiaro perchè criptati.

NOTA: Per lanciare i programmi è necessario installare le librerie di scapy e rsa.

7

Sitografia

Qui si riportano tutti i siti utilizzati durante l'implementazione del progetto:

- https://github.com/search?q=scapy+tcp+scan+python&type=repositories
- https://github.com/abhishekdeokar/sockets-with-encryption/blob/master/tcpClient.py
- https://codingcompiler.com/sockets-and-message-encryption-decryption-between-client-and-server/
- https://www.section.io/engineering-education/rsa-encryption-and-decryption-in-python/
- https://medium.com/@md.julfikar.mahmud/secure-socket-programming-in-python-d37b93233c69
- https://www.youtube.com/watch?v=U_Q1vqaJi34&t=738s
- https://it.wikipedia.org/wiki/Socket_(reti)
- https://www.youtube.com/watch?v=HKvcjsRrOds&t=807s
- https://riptutorial.com/python/topic/8710/sockets-and-message-encryption-decryption-between-client-a
- https://www.youtube.com/watch?v=YwWfKitB8aA
- https://www.youtube.com/watch?v=9BnVVLvGgoc
- https://www.thepythoncode.com/article/building-network-scanner-using-scapy?utm_content=cmp-true
- https://stackoverflow.com/questions/7541056/pinging-an-ip-range-with-scapy
- https://www.ionos.it/digitalguide/server/know-how/porte-tcp-porte-udp/
- https://it.wikipedia.org/wiki/Porte_TCP_e_UDP_standard