Report progetto Botnet Reti di Calcolatori I, Informatica
Prof. Alessio Botta, anno 2022/2023
DIETI - Università Degli Studi di Napoli Federico II

Biagio Scotto di Covella, N86003605

March 23, 2023

Contents

1 Progetto **3**

2 Implementazione **3**
2.1 File creati . . . 3
2.1.1 *Connection . . . 3
2.1.2 *FunRC . . . 4
2.1.3 *General . . . 4
2.1.4 *RemoteCommands . . . 4
2.1.5 *Drive . . . 4

3 Server **4**
3.1 Funzionalità . . . 4
3.1.1 Comando ls . . . 5
3.1.2 Comando pwd . . . 5
3.1.3 Comando cd . . . 5
3.1.4 Comando rete/network . . . 5
3.1.5 Comando filespath . . . 5
3.1.6 Comando fsearch . . . 5
3.1.7 Comando wsearch . . . 5
3.1.8 Comando find . . . 5
3.1.9 Comando info . . . 5
3.1.10 Comando download . . . 6
3.1.11 Comando file recenti . . . 6
3.1.12 Comando screenshot . . . 6
3.1.13 Comando open . . . 6
3.1.14 Comando save/salva . . . 6
3.1.15 Comando ip . . . 6
3.1.16 Comando printFile . . . 6
3.1.17 Comando help . . . 6
3.1.18 Comando clear . . . 6
3.1.19 Comando exit . . . 6
3.2 Considerazioni finali Server . . . 6

4 Client **7**
4.1 Funzionalità . . . 7
4.2 Considerazioni finali Client . . . 7

5 Running example **7**
5.1 Esempio 1: cartellaClient ('192.168.5.94', 52223) . . . 7
5.2 Esempio 2: cartellaClient ('109.115.248.9', 51828) . . . 9

6 Librerie **10**

7 Suddivisione lavoro **11**
7.1 Parte in gruppo . . . 11
7.2 Parte individuale . . . 11

8 Considerazioni finali **11**

9 Sviluppi futuri **11**

1 Progetto

L'elaborato consiste in un applicazione client/server che usa le socket, che è in grado di lavorare su macchine diverse e in cui uno dei due componenti (il bot) raccoglie informazioni sulla macchina su cui è eseguito e invia tali informazioni all'altro componente (il bot master).

2 Implementazione

La botnet implementata ha come obiettivo principale la raccolta di informazioni dal computer attaccato, garantendo al contempo stabilità e sicurezza nella comunicazione tra client e server. Il server, che funge da botmaster, accetta le connessioni e permette di navigare all'interno del computer sotto attacco per recuperare le informazioni desiderate. Il client, una volta attivato, si collega al server e fornisce le informazioni richieste e esegue i comandi che il botmaster gli invia.
La botnet è in grado di creare due tipi di connessioni: TCP e TLS/SSL. La prima è meno sicura, in quanto non garantisce la sicurezza della connessione e non cifra i dati scambiati tra client e server, mentre la seconda connessione, TLS/SSL, garantisce la sicurezza della connessione tramite l'identificazione con certificato sia del client che del server, e rende cifrati qualsiasi messaggio scambiato.
Nel corso della discussione verranno esposte tutte le funzionalità della botnet implementata.

2.1 File creati

Per semplificare la fase di debugging e rendere il codice più leggibile, si è scelto di suddividere il codice sorgente in più file, in particolare all'interno dello zip ci sono:

- client.key
- client.crt
- server.key
- server.crt
- requirements.txt

Di questi file, i primi quattro servono sia al server che al client per autentificarsi e creare una connessione TLS/SSL. Il file "requirements.txt" contiene tutte le librerie utilizzate nel codice. Infine sono lasciati i comandi di creazione dei file client.crt/.key e server.key/.crt:

- openssl req -new -newkey rsa:2048 -days 365 -nodes -x509 -keyout client.key -out client.crt
- openssl req -new -newkey rsa:2048 -days 365 -nodes -x509 -keyout server.key -out server.crt

Oltre a questi file troviamo anche:

- projectReti-Client
- projectReti-Server

Questi due file contengono tutto il codice sorgente del client e del server, ma ,essendo questo molto denso, si è deciso di suddividere ulteriormente entrambi i file come segue:

- *Connection
- *FunRC
- *General
- *RemoteCommands
- *Drive

I file sopra elencati inidicano la suddivisione del codice sia del server che del client, l'asterisco (*) indica o client o server.
Nella sezione successiva vengono spiegati nel dettaglio i file sopra elencati.

2.1.1 *Connection

Questo file, sia lato client che lato server, contiene le implementazioni di tutte le funzioni necessarie all'attivazione della connessione e alla gestione di essa. Inoltre, sono implementate due funzioni differenti a seconda se si vuole utilizzare una connessione TCP o una TLS/SSL.

2.1.2 *FunRC

Questo file, sia lato client che lato server, contiene le implementazioni di tutte le funzioni necessarie alla gestione delle richieste che il botmaster server fa al client. In particolare ogni funzione implementa una funzionalità che estrapola dati dal computer attaccato dal botmaster. Lato client queste funzioni estrapolano i dati dal computer per poi mandarli al botmaster, invece lato server gestiscono i dati che il client manda.

2.1.3 *General

Questo file, sia lato client che lato server, contiene le implementazioni di tutte le funzioni regex necessarie al controllo della corretta sintassi di tutti i dati in input. In particolare ogni funzione controlla se i comandi scritti dal potenziale "hacker" siano corretti o meno.

2.1.4 *RemoteCommands

Questo file, sia lato client che lato server, contiene le implementazioni di tutte le funzioni necessarie alla gestione dell'apertura o chiusura del remote control e all'invio del pacchetto di informazioni base che ogni client manda appena viene accettata la connessione dal server.

2.1.5 *Drive

Questo file, sia lato client che lato server, contiene le implementazioni di tutte le funzioni necessarie all'attivazione del botmaster lato server o dei thread lato client. In particolare, lato client, si notano quattro differenti thread ognuno atto alla raccolta di diversi dati quali keylogger e audio.

3 Server

Come precedentemente espresso, il server implementato svolge il ruolo di botmaster. Dopo l'avvio del programma, il server stabilirà una connessione e rimarrà in attesa di client che desiderano connettersi. Al momento in cui un client si connette al server, una serie di funzioni vengono avviate. In primo luogo, viene creata una cartella specifica ogni volta che un client si connette, dove verranno salvati tutti i risultati ottenuti dall'esecuzione dei comandi da parte del client. Inoltre, il client invia un pacchetto di informazioni che vengono stampate a video e salvate nel fileLog. Successivamente, viene attivata la funzione di remoteControl che consente al botmaster di ricevere in input i comandi da inoltrare al client. Il client eseguirà questi comandi e restituirà i relativi risultati. Il server riceve i dati dal client, li elabora e li salva nel fileLog o, a seconda delle funzioni, in file specifici. Infine, terminata l'esecuzione di tutte le azioni sul client, prima di chiudere la connessione, il client invierà tutti gli audio registrati durante l'esecuzione e i risultati ottenuti dal keyLogging. Quest'ultimi verranno salvati i specifici file. Scaricati tutti questi file, si chiuderà la connessione e, se si desidera, il server resterà in ascolto di nuove connessioni. Infine il server è implementato in modo tale che ogni azione eseguita è salvata in una variabile fileLog, la quale è salvata, dopo la chiusura della connessione, in un file.txt. In questo modo teniamo traccia di ogni azione svolta dal server sul client.

3.1 Funzionalità

```
01-List of files in a path.....................................: ls <path>
02-Current Working Directory...................................: pwd
03-Change Working Directory....................................: cd <path>
04-Network configuration.......................................: rete/network
05-Lists all files with extension (.pdf, .docx, .txt, etc...): filespath <.extension>
06-Search if a file has that word in its name..................: fsearch <"word">
07-Search if the text of a file has that word in it...........: wsearch <"word"> <path>
08-Search for an extension type in a path.....................: find <.exstension> <path>
09-Information S.O. client.....................................: info
10-Download file...............................................: download <"filename.exst"> <path>
11-Recently modified files list................................: file recenti
12-Take screenshots............................................: screenshot
13-Open a file.zip.............................................: open <"nomeFile.zip"> <path>
14-Save everything done up to that point.......................: save/salva
15-Track ip address............................................: ip
16-Read file content...........................................: printFile <"filename"> <path>
17-Open list of command........................................: help
```

```
18-Clear terminal.....................................: clear
19-Exit from remote control...........................: exit

    Type of <path>:
    .        current path
    ..       indicates the path to the previous folder
    ./<path> indicates the path from the current folder to the path entered
    <path>   indicates the path
```

Come si può notare, il programma dispone di 19 diversi comandi che permettono di estrapolare informazioni dal client sotto attacco. Ogni comando viene controllato attraverso l'utilizzo di apposite espressioni regolari (regex), che, solo in caso di corrispondenza, attiveranno le relative funzioni sia lato client che lato server. È importante notare che durante l'esecuzione di alcune di queste funzioni, potrebbe essere necessario attendere diversi secondi, se non minuti, per ottenere i risultati desiderati. Ciò è dovuto all'elevato numero di operazioni che devono essere eseguite. Durante questo periodo, il programma potrebbe sembrare bloccato, ma è sufficiente attendere e avere pazienza, poiché si tratta semplicemente di un ritardo dovuto al processamento delle richieste. Tra le funzioni implementate, abbiamo:

3.1.1 Comando ls

Il comando "ls <path>" lista tutti i file e tutte le cartelle presenti nel path desiderato.

3.1.2 Comando pwd

Il comando "pwd" ci dice qual è la working directory corrente.

3.1.3 Comando cd

Il comando "cd <path>" ci permette di cambiare working directory.

3.1.4 Comando rete/network

Il comando "rete/netwrok" ci ritorna il risultato del comando "ifconfig/ipconfig".

3.1.5 Comando filespath

Il comando "filespath <.extension>" crea un file .txt con all'interno tutti i file presenti nel file system che contengano quella determinata estensione e il percorso (path) in cui sono contenuti:

```
"lb2.pdf" nel percorso: C:\Users\biagi\AppData\Local\Programs\MiKTeX\doc\latex\base
```

3.1.6 Comando fsearch

Il comando "fsearch <"word">" setaccia il file system alla ricerca di file che contengano nel nome la parola "word".

3.1.7 Comando wsearch

Il comando "wsearch <"word"> <path>" setaccia il path desiderato alla ricerca di file che contengano al loro interno la parola "word".

3.1.8 Comando find

Il comando "find <.exstension> <Path>" è simile a filespath ma si limita a cercare i file in un determinato path.

3.1.9 Comando info

Il comando "Info" ritorna una serie di informazioni sul sistema operativo del client e anche altre informazioni sul file system.

3.1.10 Comando download

Il comando "download <"filename.exst"> <path>" scarica il file desiderato presente nel path indicato, in particolare è possibile copiare e incollare una riga del file filespath.txt:

```
download "lb2.pdf" nel percorso: C:\Users\biagi\AppData\Local\Programs\MiKTeX\doc\latex\base

File lb2.pdf successfully downloaded
```

3.1.11 Comando file recenti

Il comando "file recenti" ritorna tutti i file recentemente modificati relativi alla working directory.

3.1.12 Comando screenshot

Il comando "screenshot" effettua uno screenshot dello schermo del client.

3.1.13 Comando open

Il comando "open <"nomeFile.zip"> <path>" ritorna i file contenuti in un file .zip.

3.1.14 Comando save/salva

Il comando "save/salva" scrive su un file i risultati contenuti in quel momento nel fileLog.

3.1.15 Comando ip

Il comando "ip" ritorna tutte le informazioni relative all'indirizzo ip a cui è collegata la macchina client.

3.1.16 Comando printFile

Il comando "printFile <"filename"> <path>" stampa a video il contenuto del file richiesto. A differenza della download non scarica il file, ma stampa a video il suo contenuto.

3.1.17 Comando help

Il comando "help" stampa la lista di comandi dispobibili.

3.1.18 Comando clear

Il comando "clear" ripulisce il terminale.

3.1.19 Comando exit

Il comando "exit" chiude la fase di remoteControl.

3.2 Considerazioni finali Server

Il codice del botmaster, o del server, è stato implementato in modo tale da evitare e gestire il maggior numero di eventi indesiderati. Sia lato client che lato server, il codice è ben strutturato e gestisce correttamente eventuali problemi che possono sorgere durante l'esecuzione del software.
Per garantire una corretta sincronizzazione tra una send e una recv, si è deciso di inserire dei time.sleep() per evitare l'accavallarsi di più send e più recv. Inoltre, sono state inserite varie funzioni di abbellimento per rendere più chiaro e semplice l'uso della botnet e per tenere sempre traccia dello stato del programma e degli eventuali errori riscontrati.
Infine, il server è stato implementato in modo da gestire equivalentemente la connessione di un client con sistema operativo Linux, Windows o MacOS. Questa caratteristica garantisce la massima compatibilità con i dispositivi dei client e rende la botnet utilizzabile su un'ampia gamma di sistemi.

4 Client

Il client rappresenta, nel nostro quadro, la macchina da cui vogliamo recuperare informazioni. Il client è implementato con l'uso di quattro thread che ci aiutano a svolgere parallelamente differenti funzioni principali. In particolare abbiamo i seguenti thread:

- threadTrojan
- threadRemoteControl
- threadKey
- threadAscolto

Il "threadTrojan" funge da maschera, e mostra all'utente che sta usando il computer sotto attacco un finto task manager. Vengono stampati a video dati relativi alla macchina e tutti i processi in esecuzione proprio come un task manager. Quest'ultima funzionalità è implementata solo per windows.
Il "threadRemoteControl" si occupa di far partire il cuore principale di tutto il sorgente, ossia la parte di connessione al server e di gestione dei comandi e delle richieste di quest'ultimo.
Il "threadKey" avvia la funzione di keyLogging, ossia la funzione che tiene traccia di tutti i tasti digitati dall'utente durante la fase di attacco. In particolare tutti gli eventi da tastiera vengono salvati in una variabile che verrà poi inviata al server per essere salvata in un file.
Il "threadAscolto" è simile al threadKey ma, a differenza di questo, invece di registrare i dati da tastiera, registra l'audio ambientale captato nelle vicinannze del computer client. Anche qui i dati sono salvati in una variabile che verrà mandata poi al server.
Quando il client viene avviato i thread vengono lanciati e iniziano a lavorare in parallelo. Ciò ci permette di raccogliere tutte le informazioni desiderate mostrando all'utente del pc client una schermata fittizia.

4.1 Funzionalità

Le funzionalità implementate sono esattamente le medesime, le differenze sono dovute sono al differente modo di operare.

4.2 Considerazioni finali Client

Il codice del client è stato implementato in modo tale da gestire ed evitare il maggior numero di eventi indesiderati. Sono stati inclusi opportuni frammenti di codice per gestire qualsiasi problema che possa sorgere durante l'esecuzione del software. In caso di caduta della connessione, il client è in grado di continuare il suo funzionamento e di riprendere le operazioni non appena la connessione viene ripristinata.
Inoltre, il client è stato implementato in modo da gestire equivalentemente la connessione di un client con sistema operativo Linux, Windows o MacOS. Ciò garantisce che il software sia compatibile con un'ampia varietà di dispositivi e sistemi operativi.
Infine, nel caso in cui dovesse verificarsi una chiusura di connessione inaspettata, il client è in grado di terminare le azioni in corso e di ricollegarsi al server non appena questo torna attivo. Questa funzionalità permette di garantire un flusso costante di comunicazione e di assicurare che il client sia sempre disponibile per le operazioni successive.

5 Running example

In questa sezione è portato un esempio per mostrare il funzionamento della botnet.

5.1 Esempio 1: cartellaClient ('192.168.5.94', 52223)

Nella cartella dell'esempio 1 sono portati tutti i risultati ottenuti dal client. In particolare è possibile notare il "fileLogGenerale" che tiene traccia di tutto ciò che è stato fatto durante il funzionamento del bot sulla macchina attaccata. Inoltre ci sono altri file che contengono vari dati scaricati dal client, in particolare:

- searchFile.txt
- searchWord.txt
- filespath.txt

- funHackIp.txt
- foto.png
- fileKeyLog.txt
- fileLog*.txt

Oltre a ciò sono presenti tutti i file audio da 20 secondi scaricati dal client che contengono tutto l'audio catturato in prossimità del computer attaccato durante la fase di attacco. Infine sono presenti due file .pdf scaricati dalla macchina client.
A titolo di esempio si mostra parte del contenuto del fileLogGenerale:

```
###          FILELOG RESULT          ###

Client connected: 192.168.5.94:52223
...
[CONNECTED] Established a connection with the Victim using socket: ('192.168.5.94', 52223)
[CONNECTED] Established a connection with the Victim; Socket: ('192.168.5.94', 52223)is
connected to the server

Information on the victim's Operating System ('192.168.5.94', 52223):

Operating System: Windows
Machine: AMD64
Host: LAPTOP-JKLTQPAK
Processor: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
Platform: Windows-10-10.0.22621-SP0
Release: 10
Path: C:\Users\biagi\Desktop\Python Project\Progetto Reti Biagio\Source Code Unione

[DONE] Info received.

[REMOTE CONTROL] Procedure activated; you are now on the victim's pc in the path below...

C:\Users\biagi\Desktop\Python Project\Progetto Reti Biagio\Source Code Unione$ help

C:\Users\biagi\Desktop\Python Project\Progetto Reti Biagio\Source Code Unione$ ls .

-: cartellaClient ('192.168.5.94', 52223)
-: client.crt
-: client.key
-: projectReti-CLIENT.py
-: projectReti-SERVER.py
-: requirements.txt
-: server.crt
-: server.key


C:\Users\biagi\Desktop\Python Project\Progetto Reti Biagio\Source Code Unione$ ls ..

-: .idea
-: prova tls vs tcp.txt
-: provaBotnet13-12-22
-: Report
-: Source Code
-: Source Code Unione
-: todo.txt

...

C:\Users\biagi\Desktop\Python Project\Progetto Reti Biagio\Source Code Unione$ cd ..

C:\Users\biagi\Desktop\Python Project\Progetto Reti Biagio$ cd ..
```

```
C:\Users\biagi\Desktop\Python Project$ cd ..

C:\Users\biagi\Desktop$ cd ..

C:\Users\biagi$ cd ..

C:\Users\biagi\desktop$ salva

C:\Users\biagi\desktop$ wsearch "napoli" .


Parola chiave 'napoli' trovata 1 volte nel file:
"Progetti.docx" nel percorso: .

Parola chiave 'napoli' trovata 19 volte nel file:
"Reti Botta.pdf" nel percorso: .

----Trovate 20 elementi simili a 'napoli'.

File successfully created!

C:\Users\biagi\desktop$ download "OKL06.pdf" nel percorso: C:\Users\biagi\Desktop\PDF_LEZIONI

File OKL06.pdf successfully downloaded

C:\Users\biagi\desktop$ salva

C:\Users\biagi\desktop$ help

C:\Users\biagi\desktop$ exit

[REMOTE CONTROL CLOSED] Remote Control procedure successfully closed!

[CLOSED] Client Connection ('192.168.5.94', 52223) closed succesfully!
```

5.2 Esempio 2: cartellaClient ('109.115.248.9', 51828)

Nella cartella dell'esempio 2 sono presenti tutti i risultati ottenuti dal client e sono simili a quelli del primo esempio. La principale differenza è nella connessione utilizzata. Infatti, in questo caso, la connessione è ottenuta attraverso il principio del port forwarding, ovvero la tecnica che permette il trasferimento dei dati da un computer ad un altro tramite una specifica porta di comunicazione. A differenza dell'esempio 1, il client ha un indirizzo IP pubblico invece di un indirizzo IP privato. L'indirizzo IP "109.115.248.9" è un indirizzo IP pubblico, cioè un indirizzo univoco assegnato a una macchina connessa direttamente a Internet. Questo tipo di indirizzo IP può essere utilizzato per accedere a risorse in remoto su Internet, nel nostro caso il server. D'altra parte, l'indirizzo IP "192.168.5.94" è un indirizzo IP privato, cioè un indirizzo utilizzato all'interno di una rete locale, come ad esempio una rete domestica o aziendale. Questo tipo di indirizzo IP non è accessibile dall'esterno della rete locale e viene utilizzato per identificare i dispositivi all'interno della rete stessa.

```
    ###          FILELOG RESULT          ###

[SSL established] Peer: ...
[CONNECTED] Established a connection with the Victim using socket: ('109.115.248.9', 51828)

[CONNECTED] Established a connection with the Victim; Socket: ('109.115.248.9', 51828) is
connected to the server

Information on the victim's Operating System ('109.115.248.9', 51828):

...
```

6 Librerie

Nella corrente sezione sono riportate tutte le librerie utilizzate lato client e lato server per implementare le funzionalità sopra spiegate.
Lato server:

```
import pickle
import platform
import re
import signal
import sys
import os
import time
import traceback
import subprocess
import socket
import ssl
import soundfile as sf
import ipinfo

from tqdm import tqdm
from colorama import Fore
from time import sleep
from socket import *
from os import system
from cryptography.fernet import Fernet
```

Lato client:

```
import time
import subprocess
import os
import sys
import zipfile
import traceback
import sqlite3
import win32crypt
import ipinfo
import json
import base64
import shutil
import ssl
import pyautogui
import pickle
import psutil
import sounddevice as sd
import soundfile as sf
import keyboard  # for keylogs

from colorama import Fore
from docx import Document
from pathlib import Path
from PyPDF2 import PdfFileReader
from cryptography.fernet import Fernet
from scapy.all import *
from datetime import datetime, timedelta
from Crypto.Cipher import AES # pip install pycryptodome
from os import system
from socket import *
from wavio import write
from threading import Thread
from threading import Timer
import socket
```

7 Suddivisione lavoro

La botnet presentata è stata sviluppata in parte grazie alla collaborazione tra Scotto di Covella Biagio (N86003605), Prosciutto Erasmo (N86003546) e Lanuto Antonio (N86003762), e in parte individualmente da Scotto di Covella Biagio.

7.1 Parte in gruppo

Nella fase di lavoro di gruppo sono state implementate le funzioni di regex e la parte di connessione TCP, in più sono state implementate le funzioni "filespath", "find", "download", "remoteControl", "printInformazioni" e la funzione "trojanBehaviour".

7.2 Parte individuale

Nella fase di lavoro individuale è stata sviluppata tutta la parte di connessione TSL/SSL con le relative funzioni e le restanti funzioni di ricerca informazioni come "searchWord", "searchFile", "funHackIp", "printFile", "openZip", "commandsHelp", "decision" e altre meno importanti. In più sono stati implementati i due thread di "keyLog", e le relative funzioni, e "ascolto".

8 Considerazioni finali

Lo sviluppo del progetto mi ha permesso di approfondire la conoscenza di un linguaggio di programmazione come Python che non avevo mai utilizzato. Inoltre, è stato fonte di stimolo per cercare di implementare funzioni sempre più specifiche per il recupero di informazioni nascoste e per creare funzioni che potessero ottenere risultati specifici in breve tempo e con pochi passaggi.
Durante lo sviluppo, le principali problematiche incontrate sono state legate alla comunicazione tra due computer attraverso una connessione con le socket e alla gestione degli eventuali errori e ritardi di invio e ricezione dei messaggi. Per risolvere tali problemi, sono state fatte molteplici prove e sono state utilizzate diverse strategie.
Lo sviluppo di questo progetto ha rappresentato una sfida molto stimolante in quanto mi ha permesso di approcciarmi al campo dell'hacking e di comprendere meglio come i sistemi informatici siano vulnerabili e come sia possibile reperire informazioni sensibili con poche funzioni.

9 Sviluppi futuri

Nell'ambito di uno studio sui possibili sviluppi della tecnologia informatica, si ipotizza che in futuro sia possibile implementare funzioni ancora più specifiche all'interno di un software, consentendo l'accesso alla webcam del computer e il recupero in tempo reale di fotografie dell'utente in uso. Inoltre, si ritiene possibile l'inserimento di un trojan che possa simulare un gioco ma contemporaneamente recuperare le informazioni desiderate.
È importante sottolineare che queste funzioni non includono la possibilità di modificare il computer dell'utente, come richiesto, ma possono essere facilmente integrate in futuro sviluppi del software.