

Betriebssystem & Rechnernetze - Dokumentation

- Projekt: Alternative 4 - Passwort-Manager
- Dozent: Prof.Dr. Christian Baun
- Gruppenmitglieder
 - *Sarvar Tojikulov*
 - *Gabriel Kupresanin*
- Abgabefirst: 28.06.2021
- Quelle / WebSeite : www.christianbaun.de



Inhaltsverzeichnis

1. Allgemein

2. Installation

3. Anforderungen vom Passwort - Manager

3.1 Speicherung von Titeln, Benutzername & Passwörtern

3.2 Löschen von Einträgen nach interaktiver Bestätigung

3.3 Sichere Speicherung der Daten im Dateisystem

3.4 Automatische Generierung der Passwörter

3.5 Die Sicherung des Passwort-Manager durch ein Master-Passwort und Änderungsmöglichkeit und das Master - Passwort wird nicht im Klartext angezeigt

3.6 Das Zeitlimit für das Master - Passwort festlegen

3.7 Passwörter werden nicht im Klartext angezeigt & es wird automatisch in der Zwischenablage (Clipboard) gespeichert.

3.8 Einträge als Tabelle in der Shell ausgeben

3.9 *Eigene Funktionen*

3.9.1 Info-Befehl

3.9.2. Master - Passwort - Zusatzoption

4. Kommandozeilen-Befehl anstatt das starten eines Skripts

1. Allgemein

Die Verwendung eines Passwort-Managers soll dem Benutzer bei der Verwaltung seiner Passwörter behilflich sein, sowie die Sicherheit und den Schutz erhöhen.

Dies geschieht beispielsweise durch Verschlüsselung der Passwörter oder anderer Funktionen die der Passwort - Manager enthält.

Der Passwort-Manager kann als ein Tool beziehungsweise als eine Anwendungssoftware bezeichnet werden.

Unser Passwort-Manager kann automatisch komplexe Passwörter generieren durch Benutzerangaben, dies verhindert eventuell einfache Passwörter, die leicht zu bekommen wären wie 1234 oder das Geburtsdatum. Ein Tool wie Hydra in Kali Linux könnte sehr schnell solch ein Passwort hacken im andern Fall wie dem Geburtsdatum wäre Social Engineering eine Option um an das Passwort zugegangen.

Des Weiteren besitzt der Passwort-Manager ein sogenanntes Master-Passwort , was als eine zusätzliche Absicherung dienen soll.

2.Installation

Die **Grundvoraussetzungen** für die Installation um mit dem Password-Manager arbeiten zu können sind:

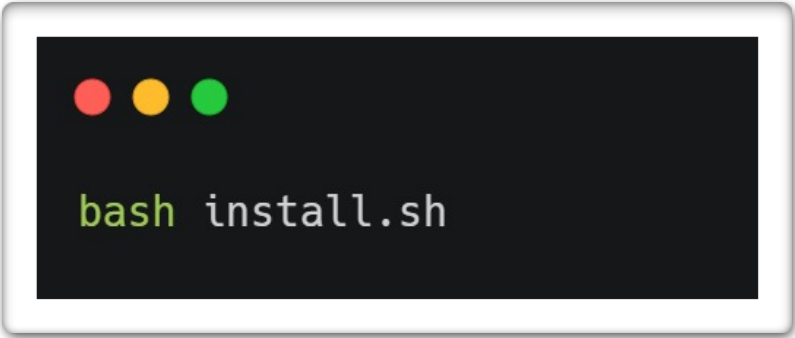
- Installation von GitBash
- Installation von Python
- Sowie pip install bcrypt

Installationsverfahren:

1. Klicken Sie [hier](#) und downloaden Sie die Datei install.sh
2. Öffnen Sie das Terminal (Bash)
3. Geben Sie das Kommando: bash install.sh ein
4. Die Installation wird gestartet
5. Geben Sie im Terminal passman usage ein um die Informations-Box zu öffnen

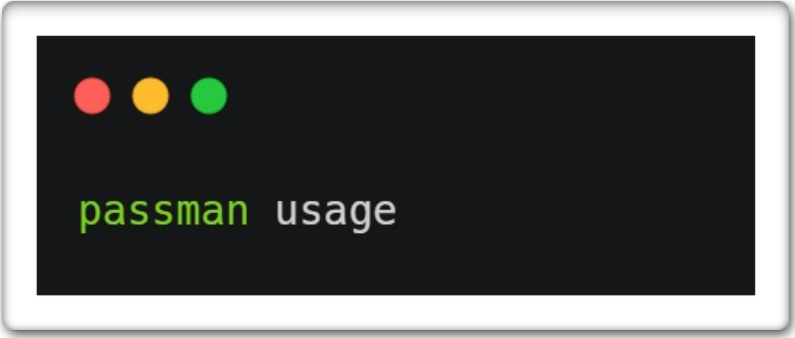
Beispiel Terminal - Eingabe:

1. Schritt



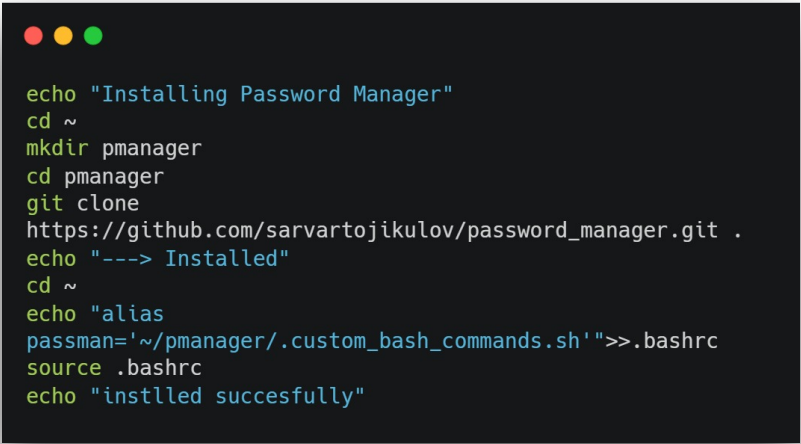
```
bash install.sh
```

2. Schritt



```
passman usage
```

- Die Eingabe von dem Befehl **bash install.sh** führt im Hintergrund folgenden Bash-Skript aus:



```
echo "Installing Password Manager"
cd ~
mkdir pmanager
cd pmanager
git clone
https://github.com/sarvartojikulov/password_manager.git .
echo "---> Installed"
cd ~
echo "alias
passman='~/pmanager/.custom_bash_commands.sh'>>.bashrc
source .bashrc
echo "instlled succesfully"
```

echo Gibt den Satz wieder Installing Password Manager.

cd ~ wechseln wir in die Home Directory

mkdir legt uns ein neues Verzeichnis an mit dem Namen pmanager

Mit **cd pmanager** gehen wir zu diesem Ordner

git clone lädt alle Daten von unserem GitHub - Projekt herunter (sieh linke)

Zum Schluss gibt uns das echo wieder das die Installation erfolgreich war.

3.1 Speicherung von Titeln, Benutzernamen & Passwörtern

Als erstens müssen wir per Terminal-Eingabe einen Usernamen erstellen und einen neuen Title anlegen. Das Passwort wird automatisch generiert darum findet die Eingabe des Passwortes hier noch nicht statt.

```
passman add -title facebook -username paul222
```

Mit diesem Python-Programm öffnen wir die JSON Datei und lesen sie mit mode "r" was für read steht, danach ändern wir JSON-String in Python-Dictionary Datentyp und speichern dies unter json_obj.

```
a_file = open(home_path + "\pmanager\db.json", "r")
json_obj = json.load(a_file)
```

Die Speicherung der Daten findet mit der Funktion **write_to_db()**: statt.

a_file öffnet dabei das angegebene Verzeichnis zum schreiben mit "w" was für write steht, dieses w ist ein sogenannter mode. Json.dump konvertiert json_obj und a_file zum JSON - String weil Python nicht mit Bit-Strings arbeitet.

Am ende wird das ganze wie ein Scanner in Java beendet mit dem Befehl **a_file.close()**.

```
def write_to_db():
    a_file = open(home_path + "\pmanager\db.json", "w")
    json.dump(json_obj, a_file)
    a_file.close()
```

Die Terminaleingabe vom Titel und Username werden hier mit der Funktion **create_obj** erstellt die ebenfalls das automatisch generierte Passwort beinhaltet.

```
def create_obj(title, username, password):
    acc_obj = {
        "title": title,
        "username" : username,
        "password" : password
    }
    return acc_obj
```

○ Zusammenfassung der Speicherung:

Mit der Funktion `create_obj` wird ein Objekt erstellt das in der `json_obj` Variable ergänzt wird und mit `write_to_db` werden alle Änderungen u.ä in der JSON - Datei abgespeichert.

3.2 Löschen von Einträgen nach interaktiver Bestätigung

Das Löschen von Einträgen haben wir mit einer **Try-Except Methode** realisiert. Der Benutzer wird zuerst nach einem Input gebeten also eine interaktive Bestätigung das der wirklich den Eintrag löschen will, dafür soll dies mit einem y bestätigen. Dies wird in der `best` Variable gespeichert.

Ist kein Titel mit der Eingabe des Nutzers vorhanden bzw. kein Titel mit solch einem Namen der existiert, so kommen wir zu **except**, dann zeigt das Programm: das die Eingabe falsch ist und wir **passman usage** eingeben sollen.

```
def eintrag_loeschen():
    best = input("Bitte bestätige, dass du den Ertrag löschen willst. (Y/N)")
    if best.lower() == 'y':
        try:
            title = sys.argv[1]
            print("Ertrag mit Titel: " + title + " | wird gelöscht")
            for elem in array:
                if elem['title'] == title:
                    json_obj['pwrds'].pop(array.index(elem))
                    write_to_db()
            print("Eintrag wurde erfolgreich gelöscht")
        except:
            print("Die Eingabe ist falsch. Bitte geben sie 'passman usage' ein.")
```

3.3 Sichere Speicherung der Daten im Dateisystem

```
{
  "mpw": {
    "pw": string,
    "salt": string,
    "mode" : string | null,
    "minutes" : number | null,
    "date" : timestamp,
    "generator" : {
      "len": number,
      "gb": boolean,
      "kb": boolean,
      "zh": boolean,
      "zf": boolean
    }
  },
  "pwrds": [
    {
      "title": string,
      "username" : string,
      "password" : string
    },
    {
      "title": string,
      "username" : string,
      "password" : string
    },
    {
      "title": string,
      "username" : string,
      "password" : string
    }
  ]
}
```

Der **JSON - File** ist unser **Dateisystem**. Es beinhaltet 2 große Objekte

dabei steht "**mpw**" für die Master-Passwort Daten und **pwrds** für alle Passwörter bzw. Einträge wie Username und Title.

Alle Angaben werden durch **write_to_database** Funktion gespeichert.

Die Abkürzung gb steht bspw. für Großbuchstaben , kb für Kleinbuchstaben, das ganze wird mit einem **boolean** gecheckt (true oder false).

3.4 Automatische Generierung der Passwörter

```
def generate_passwordList():
    password_list = []
    mode = json_obj['mpw']['generator']
    lenght = mode['len']
    for key in mode:
        if key == 'gb' and mode[key] == True:
            for char in range(65, 91):
                password_list.append(char)
        if key == 'kb' and mode[key] == True:
            for char in range(97, 123):
                password_list.append(char)
        if key == 'zh' and mode[key] == True:
            for char in range(33, 48):
                password_list.append(char)
            for char in range(58, 97):
                password_list.append(char)
            for char in range(123, 127):
                password_list.append(char)
        if key == 'zf' and mode[key] == True:
            for char in range(48, 58):
                password_list.append(char)
    return {
        "list" : password_list,
        "leange": lenght
    }
```

Mit der Funktion **generate_passwordList** kann der Nutzer des Password-Managers sein Passwort nach seinen belieben generieren lassen.

Der User wird bei der Erstellung des Accounts vorher gefragt welche Länge sein Passwort haben soll , ob es Großbuchstaben , Kleinbuchstaben , Zahlen oder sogar Zeichen enthalten soll.

Ist dies in diesem Fall bspw. **== True:** also es trifft zu das der User Großbuchstaben in seinem Passwort haben möchte so wird dies mit diesem Code realisiert.

Wir haben hierbei mit ASCII gearbeitet und Großbuchstaben , Kleinbuchstaben etc. Dabei unterteilt , bspw sind Zahlen zwischen 33 und 48.

Es ist eine Passwortliste von ASCII-Codes aber nicht das Passwort selbst was hier generiert wird.

Die Funktion

generate_password zieht aus der Funktion

generate_passwordList den ASCII - Code und die Länge (`ascii_list` , `amount`) und wandelt diese in einen String um (**char**) und das **result** ist in diesem Fall ist das generierte Passwort aus der

generate_Password_List.

```
def generate_password(ascii_list, amount):
    result = ''
    for item in range(amount):
        index = random.randint(0, len(ascii_list) - 1)
        charCode = ascii_list[index]
        result += chr(charCode)
    return result
```

Bei der neu Erstellung eines neuen Users findet die Abfrage für das Passwort hier statt. Es wird gefragt ob Großbuchstaben , Kleinbuchstaben , Zahlen oder Symbole da Passwort beinhalten soll, sowie welche Länge das generierte Passwort besitzen soll.

```
def password_generator_sett():
    def set_pw_mode(mode, string):
        str = 'Soll Passwort ' + string + ' haben? Y/N '
        a = input(str).lower()
        if a == "y":
            json_obj['mpw']['generator'][mode] = True
        elif a == "n":
            json_obj['mpw']['generator'][mode] = False
        else:
            print("Eingabe ist falsch.")
    print("Automatisch generierte Passwort Einstellungen: \n")
    laenge = int(input("Geben sie die Laenge des Passworts: "))
    json_obj['mpw']['generator']['len'] = laenge;
    set_pw_mode('gb', 'große Buchstaben')
    set_pw_mode('kb', 'kleine Buchstaben')
    set_pw_mode('zh', 'Zeichen')
    set_pw_mode('zf', 'Zahlen')
```

3.5 Die Sicherung des Passwort-Manager durch ein Master-Passwort und Änderungsmöglichkeit

```
def new_user():
    if json_obj['mpw']['pw'] == None:
        print("Du bist neu hier \nLass uns neue Master Passwort erstellen")
        if write_new_master_password() and choose_mode():
            write_to_db()
        else:
            print("Es ist ein Fehler entstanden, versuchen Sie nochmals")
```

Die Funktion **new_user** prüft zu erst ob ein User existiert oder nicht , falls nicht wird `write_new_master_password()` and `choose_mode()` ausgeführt das ganze wird mit `write_to_db` gespeichert , wenn die beiden erfolgreich ausgeführt wurden falls nicht siehe else - Anweisung.

```
def write_new_master_password():
    pw = getpass("Neue Master Passowrd: ")
    password = getpass("Password wiederholen: ")
    if(pw == password):
        salt = bcrypt.hashpw(bytes(password, encoding='utf-8'), bcrypt.gensalt())
        pw1 = bcrypt.hashpw(bytes(password, encoding='utf-8'), salt)
        json_obj['mpw']['pw'] = pw1.decode()
        json_obj['mpw']['salt'] = salt.decode()
        print("Dein Muster Passwort ist erfolgreich erstellt!")
        return True
    else:
        print("Passwörter sind nicht gleich. Versuch Nochmals")
        write_new_master_password()
```

Die Funktion **write_new_master_password** fragt bei pw nach einen Input bzw. nach einem neuen Master-Passwort, danach wird verlangt das Master-Passwort noch einmal einzugeben, wenn diese Eingaben übereinander stimmen wird es akzeptiert und verschlüsselt mit **bcrypt**.

Json_obj wird entschlüsselt mit **decode** die Funktionen dafür sind jeweils **pw1.decode()** , **salt.decode()** weil JSON-File bit-Stings nicht akzeptiert.

Als Ausgabe erhält der User das Master-Passwort das erfolgreich erstellt wurde, wenn die Passwörter am Anfang der Eingabe nicht übereinander gestimmt haben so bekommt der User die Ausgabe (Passwörter sind nicht gleich. Versuchen Sie es Nochmal).

Die Funktion wird dann mit **wirte_new_master_password()** neu gestartet.

Durch `getpass` wird das Passwort nicht im Klartext angezeigt, sondern ist unsichtbar.

3.6 Das Zeitlimit für das Master - Passwort festlegen

```
def choose_mode():
    def get_minutes():
        grenze = int(input('Wie viel Minuten: '))
        if(grenze >= 5 and grenze <= 1440):
            json_obj['mpw']['minutes'] = grenze
        else:
            print("Die Zeitangabe ist falsch. Bitte geben Sie in Zahlen zwischen 5
            und 1440 Minuten ein")
            get_minutes()
        print("--> Wie oft soll Master-Passwort abgefragt werden? Wählen sie
        Nummer. Minimum 5 Minuten, Maximum 24 Stunden (1440 min)")
        print('1. In minuten \n2. Bei jeder Abfrage')
        mode = int(input("Nummer: "))
        if mode == 1 or mode == 2:
            if mode == 1:
                json_obj['mpw']['mode'] = 'minutes'
                get_minutes()
                json_obj['mpw']['date'] = datetime.timestamp(datetime.now())
            elif mode == 2:
                json_obj['mpw']['mode'] = None
            print("Alle Eingaben sind gespeichert und verschlüssert")
            password_generator_sett()
            return True
        else:
            print('--> Unbekannte Input. Versuchen Sie nochmal')
            choose_mode()
```

Die Funktion **choose_mode** beinhaltet die Funktion **get_minutes()**: die das zeitliche Limit für die Master-Passwort festlegt. Wir haben in diesem Fall mit Minuten gearbeitet. Die Grenzen sind 5min - 24h. Mode 2 ist eine Zusatzfunktion auf die wir im Kapitel 3.9.2 eingehen. Wählt der User Mode 1 aus so soll er die Minuten angeben ab wann das Master-Passwort wieder abgefragt werden soll. Ist die Zeitangabe falsch so wird der User um eine neue Eingabe gebeten.

3.7 Passwörter werden nicht im Klartext angezeigt & es wird automatisch in der Zwischenablage (Clipboard) gespeichert.

```
def clip(password):
    pyperclip.copy(password)
    print("--> You have 30 sek till i empty the clipboard")
    signal(SIGINT, handler)
    time.sleep(30)
    pyperclip.copy(' ')
```

Die Funktion **clip** kopiert das Passwort und gibt dem User bescheid das er 30 Sekunden hat bis das clipboard auf empty bzw. auf ein Leere String gesetzt wird, dafür benutzen wir **pyperclip**. Die Funktion **signal** erkennt ob strg + c bereits benutzt wurde oder nicht, ist es ist der Fall das es benutzt wurde so kommt der leere String zum Zug. Durch das kopieren des Passwortes im Clipboard wird es nicht im Klartext ausgegeben.

3.8 Einträge als Tabelle in der Shell ausgeben

```
main_array = []
for elem in array:
    main_array.append([elem['title'],elem['username']])
print ("{:<10} {:<10} {:<10}".format("Titel      |", "Username |", "Passwort
|"))
for item in main_array:
    titel, username = item
    print ("{:<10} {:<10} {:<10}".format(titel,username, "*****"))
```

Die Einträge werden in einen Array gespeichert siehe Variable **main_array** diese Elemente aus **main_array** werden formatiert ausgegeben in einer Tabelle. Durch die ({:<10}) erhalten wir 10 Leertasten Zeichen und durch die for-Schleife werden alle Arrays im **main_array** in einer geordneten Tabelle ausgegeben.

3.9 Eigene Funktionen

○ 3.9.1 Info-Befehl

Unsere eigene erste sinnvolle Funktion die wir implementiert haben ist eine sogenannter Info-Befehl wo der User verschieden Informationen bekommt wie er einen neuen Titel und Usernamen hinzufügen kann oder einen Usernamen identifizieren kann usw.

```
usage () {  
    echo "Info: "  
    echo "    passman add:      Neue Titel und Username hinzufügen."  
    echo " "  
    echo "                -title: Flag, um den Title zu identifizieren"  
    echo "                -username: Flag, um den Username zu identifizieren"  
    echo "----> Beispiel: passman add -title Facebook -username Karl1990"  
    echo " "  
    echo "    passman copy:      Passwort mit Titel kopieren"  
    echo " "  
    echo "                -title: Flag, um den Title zu identifizieren"  
    echo "----> Beispiel: passman copy -title Facebook"  
    echo "    passman edit:      Einstellungen"  
    echo " "  
    echo "                -title: Flag, um den Title zu identifizieren"  
    echo "----> Beispiel: passman edit -title Facebook"  
}
```

```
passman usage
```

Um die Informations-Box zu öffnen muss der User `passman usage` im Terminal eingeben und es erscheint die oben drüber angegebenen Informationen die durch **echo**: geprintet werden.

3.9.2. Master - Passwort - Zusatzoption

```
def choose_mode():
    def get_minutes():
        grenze = int(input('Wie viel Minuten: '))
        if(grenze >= 5 and grenze <= 1440):
            json_obj['mpw']['minutes'] = grenze
        else:
            print("Die Zeitangabe ist falsch. Bitte geben Sie in Zahlen zwischen 5
und 1440 Minuten ein")
            get_minutes()
    print("--> Wie oft soll Master-Passwort abgefragt werden? Wählen sie
Nummer. Minimum 5 Minuten, Maximum 24 Stunden (1440 min)")
    print('1. In minuten \n2. Bei jeder Abfrage')
    mode = int(input("Nummer: "))
    if mode == 1 or mode == 2:
        if mode == 1:
            json_obj['mpw']['mode'] = 'minutes'
            get_minutes()
            json_obj['mpw']['date'] = datetime.timestamp(datetime.now())
        elif mode == 2:
            json_obj['mpw']['mode'] = None
    print("Alle Eingaben sind gespeichert und verschlüssert")
    password_generator_sett()
    return True
else:
    print('---> Unbekannte Input. Versuchen Sie nochmal')
    choose_mode()
```

In der Zeile **elif mode == 2** befindet sich eine Zusatzoption für das Master-Passwort. Gibt der User eine 2 ein so wird das Master-Passwort bei jeder Abfrage angefordert. Dies ist besonders nützlich wenn wir mit sehr vertraulichen Daten arbeiten, oder wir keine feste Zeit im Moment angeben können.

4. Kommandozeilen-Befehl anstatt das starten eines Skripts

```
.bashrc File:
```

```
alias passman='~/pmanager/.custom_bash_commands.sh'
```

Der alias ist ein in Shell integrierter Befehlsaufruf mit dem wir den Skript anhand des Terminals dem Kommando-Befehl **passman** ausführen können.

```
.custom_bash_commands.sh File:
```

```
if [[ -z $1 ]]
then
    usage
else
    winpty python ~/pmanager/main.py
fi

ITER=0
for item in $@
do
    case $item in
        -title)
            index=$(expr $ITER + 2)
            title=${!index}
            ;;
        -username)
            index=$(expr $ITER + 2)
            username=${!index}
            ;;
        esac
        ITER=$(expr $ITER + 1)
    done

    if [[ $1 == "add" ]]
    then
        winpty python ~/pmanager/add.py $title $username
    elif [[ $1 == "copy" ]]
    then
        winpty python ~/pmanager/copy.py $title
    elif [[ $1 == "edit" ]]
    then
        winpty python ~/pmanager/edit.py $title
    elif [[ $1 == 'usage' ]]
    then
        usage
    fi
```

Das ist unser Haupt-Bash-Skript wo die Möglichkeiten add , copy und usage verzeichnet sind und mit -title , - username flags die Attribute aussortieren und weiter mit Python Programm ausgeführt werden können.