Dokumentation von Werkstück A Alternative 4 'Passwortmanager' im SS-2021

Felix Kerkmann, Kenan Yücetas, Mohamad Alsaad

Am 24. Juni 2021

Zusammenfassung Diese Dokumentation beschäftigt sich mit der Implementation eines Passwortmanagers nach Vorgaben des Werkstücks A, Alternative 4 von Prof. Dr. Christian Baun im Modul Betriebssysteme & Rechnernetze. Der Passwortmanager wurde mit der Programmiersprache Python entwickelt. Für die Dokumentation wurde LaTeXverwendet. Die Dokumentation zeigt die Funktionalitäten des Programms, sowie dessen Implementierung.

1 Einleitung

Viele Menschen nutzen täglich ihre technischen Endgeräte. Sie besuchen soziale Netzwerke um sich untereinander auszutauschen, rufen ihre E-Mails ab, streamen Videos von Youtube und Netflix oder melden sich an, um über Online-Portale zu studieren. Dies sind nur einige wenige Beispiele, bei denen die Nutzer der Dienste und Plattformen ein Nutzerkonto erstellen müssen, um die Dienste zu nutzen. Bei der Registrierung wird jedoch fast jedes Mal verlangt ein individuellen Benutzernamen und ein sicheres Passwort zu erstellen, das zudem idealerweise exklusiv nur auf dieser Plattform genutzt werden sollte.
Bei all den bei der Registrierung anfallenden Zugangsdaten kann man leicht den Überblick verlieren. Passwortmanager, -so wie hier implementiert-, können dem Nutzer das Leben signifikant erleichtern.
Sie geben dem Nutzer die Möglichkeit sein „Online-Leben“ zusätzlich abzusichern. Denn das größte Problem bei Passwörtern ist, dass man zu viele davon hat und sich diese vor allem merken muss. Gerade dieses „merken müssen“ führt zu einem der größten Sicherheitsproblemen. Das Problem liegt etwa darin, dass sich die Nutzer ihre Passwörter nicht alle merken können und oftmals auf Papier notieren und in unmittelbarer Nähe zu ihren Geräte (bspw. dem PC-Arbeitsplatz) lassen oder etwa das gleiche Passwort bei mehreren Benutzerkonten verwenden.
Dieses Verhalten, obgleich nicht böswillig, führt doch zu einem großen Sicherheitsrisiko.

Dieses Problem lösen moderne Passwortmanager, da sie belieg viele individuelle Passwörter speichern können und man diese mit **nur einem gemerktem Master-Passwort** entschlüsseln kann. Die Implementierung dieses Passwortmanagers soll dem beschriebenen Sicherheitsrisiko größtmöglich entgegenwirken.

2 Anforderungen an den Passwortmanager

Der Passwortmanager soll so implementiert werden, dass dieser von der Commandozeile aus gestartet werden kann. Ein graphisches Interface ist nicht vorgesehen.
Der Benutzer soll in der Lage sein, eine Passwort-Vault zu erstellen und in dieser beliebig viele Einträge (Passwörter) zu speichern. Die Passwort-Vault soll zusätzlich durch ein Master-Passwort vor fremdem Zugriff geschützt werden und persistent im Dateisystem gesichert werden. Das Master-Passwort soll vom Nutzer initial selbst gewählt werden können und zu jeder Zeit (-nach erfolgreicher Authentifizierung-)geändert werden können. Auch soll das Master-Passwort bei der Eingabe nicht im Klartext erscheinen. Damit die Datensicherheit jedoch gewährleistet ist, muss ein vom Benutzer festlegbarer Zeitraum gesetzt werden, zu dem der Benutzer zur Eingabe des Master-Passworts erneut aufgefodert wird.
Einmal durch das Master-Passwort authentifiziert, soll der Benutzer in der Lage sein, Passwörter zu erstellen. Das Programm soll dem Benutzer ermöglichen den Titel des Eintrags, einen Benutzernamen und ein Passwort zu erstellen und in der Vault zu speichern. Hierbei sollen Passwörter jedoch nicht im Klartext erscheinen.
Wird ein Passwort vom Benutzer abgerufen, soll dieses im Zwischenspeicher für höchstens 30 Sekunden abgelegt werden. Möchte der Benutzer alle Einträge auf einmal abrufen, so sollen ihm diese direkt in der Shell als Tabelle ausgegeben werden.
Der Benutzer soll in der Lage sein, sich ein vom Programm automatisch erstelltes Passwort generieren zu lassen. Dabei ist entscheidend, dass der Benutzer die Möglichkeit hat aus verschiedenen Kriterien zu wählen. Unter die Kritereien fallen die Wahl der Länge, der Groß-&Kleinschreibung, sowie Sonderzeichen und Symbole. Der Benutzer kann diese Kriterien entweder alle, nur vereinzelt oder gar nicht auswählen und bekommt ein entsprechendes Passwort generiert. Auch soll der Benutzer die Möglichkeit haben, einmal gespeicherte Passwörter (-nach erfolgreicher Authentifizierung-) zu jeder Zeit zu löschen. Dies soll aber erst nach einer eindeutigen Bestätigung des Benutzers erfolgen.

3 Zusätzlich implementierte Funktionen

Es ist sinnvoll, ein Programm vor fremden und ggf. bösartigen Zugriffen bestmöglich zu schützen. Das Master-Passwort schützt vor unbefugtem Zugriff, ist aber für sich nicht ausreichend. Aus diesem Grund ist eine Funktion implementiert, die überwacht, wie oft ein Passwort falsch eingegeben wird. Wird das Passwort 3 mal in Folge falsch eingegeben, beendet sich das Programm automatisch. Dies soll Angriffe, wie bspw. einen Brute-Force-Angriff deutlich erschweren.
Eine weitere Funktion ist implementiert, die eine Daten-Recovery initiiert, sollte der

Benutzer dies wünschen. Dies wird unter Umständen nötig, sollte der Benutzer aus Versehen einzelne Ordner löschen. Die Recovery-Funktion ermöglicht die Ordner Struktur erneut herzustellen und hilft bei der Vergabe eines neuen Master-Passwortes, sodass das Programm weiterhin funktionsfähig bleibt.

4 Implementierung

4.1 Authentifizierung und Initial-Vergabe des Master-Passworts

Wenn sich der Benutzer zum ersten Mal anmeldet, wird er aufgefordert ein initiales Master-Passwort zu erstellen. Das initiale Master-Passwort ist nur sichtbar während der Erst-Initiierung. Bei der späteren Abfrage, wird das Passwort nicht mehr im Klartext angezeigt.

```
1  #Anmeldefunktion bei Start des Programms
2  def anmeldung():
3  if os.path.exists("PManager") != True:
4      print("Willkommen.")
5      benutzer_password = input("Geben Sie ihr Masterpasswort ein:\t")
6  #Ausschluss von leerem Passwort oder Leerzeichen
7          if benutzer_password._contains_("") or benutzer_password =="":
8              print("Das Passwort darf nicht leer sein oder ueber
9                  Leerzeichen verfuegen.Bitte erneut veruschen. \n")
10             time.sleep(0.5)
11             anmeldung()
12         else:
13 #Erstellt Datei-Ordner, in dem Passwoerter gespeichert werden
14             make_Directory()
15             masterPassword_from_file = open(pfad + "\\PManager\\
16                 MasterPW\\MasterPW.txt","w+")
17             masterPassword_from_file.write(benutzer_password)
18             masterPassword_from_file.close()
19 # default Zeitlimit wird gespeichert
20             default_Zeit_Datei = open(pfad + "\\PManager\\
21                 Zeitlimit\\Zeitlimit.txt", "w+")
22             default_Zeit_Datei.write("5")
23             default_Zeit_Datei.close()
24 # Zeitlimit-Funktion
25             thread()
26 # Rueckkehr zu Optionen
27             optionen()
```

Listing 1: Initial-Start Vergabe des Masterpassworts Ausschnitt

Sollte sich der Benutzer zum ersten Mal anmelden, wird ein Ordner „PManager“ auf der lokalen Festplatte erstellt und unter dem Desktopverzeichnis abgespeichert. In diesem Ordner werden 3 Unterordner erstellt. Ein Unterordner heißt „MasterPW“, in welchem später das Master-Passwort als .txt-Datei gespeichert wird, der zweite Unterordner heißt

„passwords“, in diesem werden später die Passwörter gespeichert und der dritte Ordner „Zeitlimit“, in dem die Einstellung für die automatische Abmeldung gespeicht wird. All diese Ordner werden durch den Aufruf der Funktion **make_Directory** realisiert.

4.2 Spätere Anmedlung

Startet der Benutzer das Programm zum zweiten Mal, so existiert bereits ein Verzeichnis auf der Festplatt mit den Zugangsdaten des Benutzers. Möchte sich der Benutzer erneut anmelden, so verlangt das Programm das Masterpasswort beim Start. Druch Eingabe des Masterpasswortes authentifiziert sich der Benutzer und gelangt in das Options-Menü. Sollte der Benutzer 3 mal ein falsches Passwort eingeben, so beendet sich das Programm **(siehe Abschnitt 4.9).**

4.3 Optionen

Der Aufruf der Optionen lässt den Benutzer durch die Eingabe von Shortcuts auswählen, ob er einen Titel löschen **d**, einen neuen Titel hinzufügen **a**, alle Titel auflisten **t**, das Programm beenden **e** oder das Master-Passwort **c** ändern möchte. Gibt der Benutzer aus Versehen eine falsche oder nicht-existente Tastenkombination ein, gibt das Programm eine kurze Fehlermeldung aus und zeigt die Optionen erneut an. Der Quellcode für den Funktionsaufruf **Optionen** kann im Anhang gefunden werden **(Siehe Anhang 1)**

4.3.1 Option: Anzeigen

Der Benutzer kann durch Drücken der **t-Taste** alle gespeicherten Einträge in der Shell ausgeben lassen.

4.3.2 Option: Löschen

Möchte der Benutzer Einträge löschen, so gelangt er durch Drücken der **d-Taste** in das Kontext-Menü zum Löschen der Einträge. Hier kann der Benutzer unter Angabe des Eintrag-Namens aussuchen, was gelöscht wird. Bevor jedoch eine Löschung stattfindet, wird der Benutzer gebeten den gewählten Eintrag durch ein Drücken von y (für yes, dt. ja) oder n (für no, dt. nein) zu bestätigen oder den Vorgang abzubrechen. Diese Maßnahme soll Fehl-Löschungen bestmöglich vermeiden.

```
#Erst Pfad-Wechsel zu Passwoertern, dann ausgeben der Passwoerter
# aus Array durch for-Schleife

        delTitel = input("Welcher Titel soll geloescht werden?
                ('e' zum zurueckkehren)\t") +".txt"
#Auswahl des Titels
        if delTitel =="e.txt":
#Zurueck ins Menue wenn e gedrueckt wird
            optionen()
#Ueberprueft, ob Eingabe existiert
        elif os.path.exists(delTitel):
            if input("Sind Sie sicher?y/n\t").lower() == "y":
#Loescht Titel
                os.remove(delTitel)
                print("Titel erfolgreich geloescht.")
                optionen()
            else:
                optionen()
        else:
            print("Dieser Titel existiert nicht.\n")
#Erneutes Aufrufen der Funktion 'Delete()'
            Delete()
```

Listing 2: Löschfunktion Ausschnitt

4.3.3 Option: Hinzufügen

Will der Benutzer ein Passwort zu seinen Einträgen hinzufügen, so kann er dies im Options-Menü unter Drücken der **a-Taste** tun. Der Benutzer hat dann die Möglichkeit einen neuen Eintrag zu erstellen, der aus dem Titel, dem Benutzernamen und dem Passwort besteht. Letzteres kann entweder direkt vom Benutzer eingegeben werden oder alternativ vom Passwort-Manager erstellt werden. Der Benutzer wird jedoch in jedem Fall gefragt, ob er das Passwort selbt eingeben möchte oder sich eines erstellen lassen möchte. (Für die Funktionsweise der Passwort-Erstellung siehe **Abschnitt 4.4**)
Der Benutzer kann die Passwort-Erstellung jederzeit mit der **e-Taste** beenden und in das Options-Menü wechseln.

4.3.4 Option: MPW Ändern

Sollte sich der Benutzer wünschen zu einem späteren Zeitpunkt das Master-Passwort zu ändern, so kann er dies mit Drücken der **c-Taste** tun. Der Benutzer wird in ein neues Menü geführt, in welchem er aufgefordert wird, seinen Wunsch zur Änderung des Master-Passwortes erneut zu bestätigen. Drückt der Benutzer y (für ja), so wird er aufgefordert das neue Passwort einzugeben. Um zu überprüfen, dass sich der Benutzer nicht vertippt hat, fordert das Programm eine erneute Eingabe des Passwortes ein und überprüft deren

Gleichheit. Sind beide Eingaben identisch, so überschreibt das Programm das alte Master-Passwort und gibt dem Benutzer über eine entsprechende Erfolgsmeldung ein Feedback. Sollten die Passwörter nicht übereinstimmen, so bekommt der Benutzer ein entspechendes Feedback und wird erneut gebeten ein neues Passwort zu erstlellen. Dürckt der Benutzer n (für nein), so wird er zurück ins Options-Menü geleitet.

```
def change_Master_Passwort():
#erneute Bestaetigung des Users
        inp = input("Master-Passwort wirklich aendern?y/n\t")
                .lower()
        if inp == "y":
            neu = input("Neues Passwort:\t")
#doppelte Eingabe des neuen Passworts, um Schreibfehler vorzubeugen
            neu2 = input("Neues Passwort wiederholen:\t")
            if neu == neu2:
#Erneute Pruefung auf leeres Passwort (weggelassen aus Platzgruenden)
                os.chdir(pfad+"\\PManager\\MasterPW")
                w = open("MasterPW.txt", "w")
#Ueberschreiben des alten MasterPW mit dem neuen MasterPW
                w.write(neu)
                w.close()
                print("Master-Passwort erfolgreich geaendert.")
#Rueckkehr Optionen
                optionen()
            else:
                print("Eingaben stimmen nicht ueberein.")
#Erneuter Aufruf der Funktion change_Master_Passwort()
                change_Master_Passwort()
```

Listing 3: Masterpasswort ändern Ausschnitt

4.3.5 Option: Zeitlimit ändern

Der Benutzer kann das Zeitlimit festlegen, in welchem das Programm sich automatisch selbst beendet, sollte es in dieser Zeit zu keinem Input des Benutzer kommen. Dabei kann der Benutzer aus 6 Alternativen wählen. **1,5,10,30 Minuten oder 1 oder 2 Stunden** (Die Komplette Funktion ist im **Anhang 2** zu finden.)

```
def zeitlimit():
    Zeitoptionen=["a: 1 Minute","b: 5 Minuten", (..usw..)]
#Aufrufen der Datei, in der das Zeitlimit gespeichert ist
    R = open(pfad + "\\PManager\\Zeitlimit\\Zeitlimit.txt", "r")
#Anzeigen des aktuellen Zeitlimits
    print("\nAktuelles Zeitlimit: "+R.read()+" Minute(n).")
    R.close()
    benutzer_input = input("Moechten Sie ein neues Zeitlimit
                            festlegen?y/n\t").lower()
    def Limit_return(limit):        #Zeitlimit-Optionen
```

```
1 #Zeit-Optionen Ausschnit (insgesamt 6 Optionen)
2         if limit == "a":
3             return '1'
4         elif limit == "b":
5             return '5'
```

Listing 4: Ausschnitt Zeitlimit Funktion

4.3.6 Option: Programm beenden

Das Programm kann mit der **e-Taste** im dem Options-Menü jederzeit beendet werden.

4.4 Passworterstellung

Die Implementierung der Passworterstellung kann **im Quellcode in den Zeilen 415-524** eingesehen werden. Die Passworterstellung wird durch die Funktion **passwort_Erstellung** aufgerufen. Zunächst wird der Benutzer aufgefordert zu wählen, ob er ein eigenes Passwort oder ein automatisch generiertes Passwort erstellen möchte. Wählt der Benutzer ersteres aus, wird durch die **input-Funktion** auf eine Eingabe gewartet, die dann im Passwort-Ordner gespeichert wird.
Sollte der Benutzer wählen, sich ein automatisches Passwort generieren zu lassen, so wird er gefragt, wie lange das Passwort sein soll. Dies geschieht durch den Aufruf der Funktion **scannerPWLength**. Die Funktion wartet auf eine Eingabe des Benutzers, speichert den Wert der Eingabe (unter passwordLength) und gibt ein return dieses Wertes zurück. (Eine Exception springt ein bei Fehlangaben und startet die if-Schleife erneut, bis eine valide Eingabe erfolgt.)
Im nächsten Schritt wird der Benutzer gefragt, ob er Zahlen im Passwort wünscht. Diese Abfrage erfolgt durch die Funktion **scannerNumberCheck**. Hierbei handelt es sich um eine boolesche Abfrage, die abermals auf eine Eingabe des Benutzers wartet, welche (unter booleanNumber) einen return gibt.
Das gleiche Prinzip wird auch bei der Frage nach dem Wunsch nach Großbuchstaben und Sonderzeichen angewendet. Die Werte ersterer Eingabe werden unter **booleanGroßLetter** , der zweite unter **booleanSonderzeichen** zurückgegeben.
Die eigentliche Erstellung des Passworts findet mit der Funktion **password** statt. Diese kriegt alle return Werte von den obigen Implementierungen vorgegeben und erstellt das Passwort nach den Eingaben des Benutzers. Durch eine Reihe von if-Verschachtelungen wird das Passwort, je nach Wünschen des Benutzers generiert. Dabei wird ein leerer String erzeugt, in welchem durch die Methode random.choice (sequence) mit den vorinitialisierten Strings (string.ascii_uppercase + string.ascii_lowercase + string.digits + string.punctuation) ein neues Passwort geschrieben wird. Natürlich werden nur die jenigen vorinitialisierten Strings verwendet, die der Benutzer bei der Eingabe explizit ausgewählt hat. Der String mit dem neuen Passwort wird durch ein 'return' zurückgegeben und im passwords-Verzeichnis unter password.txt gespeichert.

4.5 Passwort-Check

Die Funktionsweise des Passwort-Checks kann **im Abschnitt 4.8** nachvollzogen werden. Dort ist dieser mit einer Zusatzfunktion implementiert.

4.6 Zeitlimit automatische Abmeldung

Ist der Benutzer eine gewisse Zeit inaktiv wird er aus Sicherheitsgründen automatisch abgemeldet und das Programm beendet sich. Der Benutzer muss dann erneut das Programm starten und sich mit seinem Passwort anmelden. Das Zeitlimit ist standartmäßig auf 5 Minuten eingestellt, kann aber im Optionen-Menü selbstständig (durch Drücken der **z-Taste**) angepasst werden. Der Benutzer kann die Zeit in den folgenden Intervallen anpassen: **1 Minute, 5 Minuten, 10 Minuten, 30 Minuten, 1 Stunde und 2 Stunden.**

```
1  def countdown2():
2  #Aufrufen der Datei, in der die selbstgewaehlte Minutenanzahl
3  #gespeichert ist
4          open_File = open(pfad+"\\PManager\\Zeitlimit\\
5                           Zeitlimit.txt","r")
6          min = int(open_File.read())
7          timer = min*60
8  #Timer
9          for x in range(int(timer)):
10             timer -= 1
11             time.sleep(1)
12         if timer == 0:
13             print("\nAutomatisch abgemeldet, da Zeitlimit von
14                 "+str(min)+ " Minuten erreicht.")
15 #System wird beendet, wenn timer 0 erreicht
16             os.execl(sys.executable, sys.executable, *sys.argv)
```

Listing 5: Automatische Abmeldung nach festgelegter Zeit

4.7 Zeitlimit Zwischenablage

Der Passwortmanager besitzt die Funktion Passwörter in die Zwischenablage zu kopieren. Aus Sicherheitsgründen ist es jedoch wichtig, die Zwischenablage nach einer gewissen Zeit zu leeren, um zu verhindern, dass ein Einfügen aus Versehen passiert oder andere u.U. bösartige Programme die Zwischenablage zu einem späteren Zeitpunkt auslesen können. Hierfür ist die Funktion **countdown** zuständig. Diese Funktion sichert, dass die Passwörter in der Zwischenablage nur für **30 Sekunden** gespeichert werden und danach durch einen leeren String ersetzt werden. Die Implementierung ist im **Anhang 3** zu finden.

4.8 Zusatzfunktion: Programmende bei 3 Fehlversuchen der Authentifizierung

Das Masterpasswort schützt den Zugang zu der Vault mit den Passwörtern und ist somit eines der Hauptsicherungen zum Schutz vor unbefugtem Zugriff Dritter. Es kann jedoch passieren, dass sich Fremde durch schiere Gewalt, wie bspw. durch einen Brute-Force-Angriff Zutritt verschaffen wollen. Aus diesem Grund ist diese Sicherheitsfunktion implementiert worden. Sollte ein Fremder versuchen das Masterpasswort zu erraten, so hat dieser nur **3 Versuche** bevor das Programm sich beendet. Eine Brute-Force-Attake, welche darauf setzt in kurzer Zeit tausende Kombinationen in das Passwortfeld einzugeben und so durch reines Probieren den Zugang zu erlangen, wird somit eindeutig erschwert, da das Programm jedes mal erneut gestartet werden muss und somit die Zeit gegen den Angreifer spielt.

```
1  #Masterpasswort Abfrage
2  def master_passwort_check(user_first_password, count,
3                            user_second_password_for_the_check):
4  #true wenn das eingegebene Passwort mit dem MasterPW uebereinstimmt,
5          if user_first_password==user_second_password_for_the_check:
6              return True
7  #zaehlt Fehlversuche
8          else:
9                  if count < 2 :
10                         print("Falsches Passwort. Sie haben noch",
11                                 2 - count,"Versuch(e).")
12 #erneute Abfrage des MasterPW
13                 user_first_password=getpass("Bitte erneut eingeben:\t")
14                 count += 1
15                 if master_passwort_check(user_first_password, count,
16                                          user_second_password_for_the_check):
17                         return True
18         else :
19                 print("Zu viele Fehlversuche!
20                         Das Programm wird beendet.")
21                 return False
```

Listing 6: Master-Passwort-Check und Zusatzfunktion

4.9 Zusatzfunktion: Recovery

Sollte der Programm-Ordner gelöscht werden, in welchem die Dateien abgelegt werden, so bietet das Programm einen Recovery-Modus an. Implementiert ist dieser unter **exceptionHandling.** Wählt der Benutzer den Recovery-Modus aus, wird die Ordner-Struktur auf dem Dateisystem wiederhergestellt und der Benutzer muss sich ein neues Passwort erstellen. Der Quellcode mit Kommentaren ist im Anhang beigefügt **(Anhang 4)**.

5 Schlusswort

Die Bearbeitung des Projekts hat einige Zeit in Anspruch genommen und vor allem gezeigt, wie aufwändig eine Projektarbeit sein kann. Gerade in Zeiten, wie dieser, in welchen man seine Gruppen-Partner nicht direkt kennenlernen kann, ist es eine große Herausforderung ein Projekt gemeinsam zu koordinieren und die Aufgaben aufzuteilen. Es kann jedoch gesagt werden, dass trotz dieser Hürden die Bearbeitung doch recht gut gelungen ist und die Stärken der einzelnen Gruppenmitglieder zum Tragen gekommen sind.
So hat sich die Gruppe gemeinsam entschieden das Options-Menü einzuführen, statt langer aneinandergereihter Befehle. Dies liegt vor allem daran, dass der Passwortmanager keine grafische Oberfläche besitzt und dadurch eine für den Benutzer leichter verständliche Bedienung des Programms ermöglicht wird.
Die Zusatzfunktion, welche nach 3 Fehlversuchen das Programm beendet, hätte durch die Gruppe weiter ausgebaut werden können. So wäre eine zukünftige Implementierung denkbar, in welcher nach 3 Fehlversuchen das Programm nicht direkt beendet werden würde, sondern ein Counter eingebaut werden könnte, welcher nach jeden 3 Fehlversuchen eine Erhöhung der Wartezeit (Verdopplung) bis zur nächsten Eingabe mitbringen könnte. Natürlich sind noch viele solcher Ergänzungen sinnvoll und u.U. nötig, bis ein vollständig nutzbarer Passwortmanager vorliegt. Dies ist aber zweifelslos nicht möglich in so kurzer Zeit. Auch war die Gruppe vor die Herausforderung gestellt, die Dokumentation auf nur 10 Seiten zu schreiben und damit zu entscheiden, welches die wichtigsten Punkte des Projektes sind. Insgesamt kann aber gesagt werden, dass die erste Berührung mit einem Programmier-Projekt für die Gruppe eine spannende Unternehmung war und die Kompetenzen aller Gruppenmitglieder positiv ergänzt hat.

Nachtrag
Für die Interoperabilität zwischen Windows-& Linux-Systemen wurde der Quellcode nochmals angepasst. Genauer gesagt, wurde der Quellcode noch um die Funktion **getsys()** erweitert. Diese Erweiterung erfolgte in die entsprechenden Funktionen, um zwischen den unterschiedlichen Dateisystempfaden differenzieren zu können.

A Anhang 1

```
#Anzeigen der Optionen :
def optionen():
        print("\nWas moechten Sie tun?\t")
        print("")
        for elem in options:
            print(elem)

#Input-Eingabe des Users
        user_input =input().lower()

#Titel loeschen
        if user_input == "d":
            delete()        #Aufrufen der Funktion Delete()

#Alle Titel auflisten:
        elif user_input == "t":
           showTitel()      #Aufrufen der Funktion ShowTitel()

#Hinzufuegen von Titeln:
        elif user_input == "a":
            addTitel()      #Aufrufen der Funktion AddTitel()
            optionen()      #Aufrufen der Funktion optionen()

#Programm beenden:
        elif user_input == "e":
            print("Das Programm wird beendet.")
            os.execl(sys.executable, sys.executable, *sys.argv)

#Aendern des MWP
        elif user_input == "c":
            change_Master_Passwort()    #Aufrufen der Funktion chMPW()

#Aendern des Zeitlimits:
        elif user_input =="z":
            zeitlimit()     #Aufrufen der Funktion Zeitlimit()

        else:
            print("Eingabe stimmt mit keiner Option ueberein.")
            time.sleep(2.5)  #Unterbrechung des Programms fuer 2.5 sek
            optionen()   #Aufrufen der Funktion optionen()
```

Listing 7: Aufruf der Optionen

B Anhang 2

```
#Waehlen des Zeitlimits, bis automatisch beendet werden soll
def zeitlimit():
    Zeitoptionen=["a: 1 Minute","b: 5 Minuten","c: 10 Minuten",
                  "d: 30 Minuten","e: 1 Stunde","f: 2 Stunden"]
#Aufrufen der Datei, in der das Zeitlimit gespeichert ist
    R = open(pfad + "\\PManager\\Zeitlimit\\Zeitlimit.txt", "r")
#Anzeigen des aktuellen Zeitlimits
    print("\nAktuelles Zeitlimit: "+R.read()+" Minute(n).")
    R.close()
    benutzer_input = input
        ("Moechten Sie ein neues Zeitlimit festlegen?y/n\t").lower()
#Zeitlimit-Optionen
    def Limit_return(limit):
        if limit == "a":
            return '1'
        elif limit == "b":
            return '5'
        elif limit == "c":
            return '10'
        elif limit == "d":
            return '30'
        elif limit == "e":
            return '60'
        elif limit == "f":
            return '120'
        else:
            print("Eingabe stimmt nicht ueberein.")
#ernueter Aufruf der Funktion Zeitlimit(), wenn Eingabe nicht
#mit den Oben angegebenen Optionen uebereinstimmt
            zeitlimit()
```

Listing 8: Zeitlimit-Optionen

C Anhang 3

```
1  #Timer fuer Zwischenablage
2  def countdown():
3  #Timer auf 30 Sekunden festgelegt
4          timer = 30
5  #for-Schleife, jede Sekunde wird dem Integer timer 1 abgezogen
6          for x in range(30):
7              timer -= 1
8              time.sleep(1)
9  #nach 30 Sekunden wird die Zwischenablage mit einem leeren String
10 #aktualisiert
11         if timer == 0:
12             r = Tk()
13             r.withdraw()
14             r.clipboard_clear()
15             r.clipboard_append("")
16             r.update()
```

Listing 9: Passwortspeicherung in Zwischenablage

D Anhang 4

```
1
2  def exceptionHandling():
3      eingabe = input("Moechten Sie eine Recovery durchfuehren,
4           indem Sie ein neues Password fuer das Programm erstellen
5            muessen?y/n\t")
6      if eingabe =="y":
7          os.chdir(pfad + "\\PManager")  # ändert Dateien-Pfad
8  # erstellt Ordner 'MasterPW' in 'PManager'
9          os.makedirs("MasterPW")
10         benutzer_passwort = input("Geben Sie ihr Masterpasswort ein:")
11         if benutzer_passwort.__contains__(" ") or
12                 benutzer_passwort == "":
13             print("Bitte kein leeres Passwort eingeben.")
14             anmeldung()
15         else:
16 # Erstellen der Textdatei, in welcher das MasterPW gespeichert wird
17             masterPassword_Datei = open(pfad + "\\PManager\\MasterPW\\
18                                         MasterPW.txt","w+")
19 # MasterPW wird in Textdatei gespeichert
20             masterPassword_Datei.write(benutzer_passwort)
21             masterPassword_Datei.close()
22             thread()  # Funktion, die countdown2() (Zeitlimit) startet
23             optionen()  # Rückkehr zu Optionen
24
25     elif eingabe == "n":
26         last_question = input("Sind Sie sicher? Alle Ihre Daten werden
27                                unwiderruflich geloescht.y/n\t")
28         if last_question == "y":
29             shutil.rmtree(pfad + "\\PManager")
30             print("Alle Daten gelöscht. Das Programm wird beendet.")
31             os.execl(sys.executable, sys.executable, *sys.argv)
32
33         elif last_question == "n":
34             exceptionHandling()
35         else:
36             print("Falsche Eingabe.")
37             time.sleep(1)
38             exceptionHandling()
39
40     else:
41         print("Falsche eingabe! Bitte richtig eingeben!!")
42         time.sleep(2)
43         exceptionHandling()
```

Listing 10: Recovery-Funktion