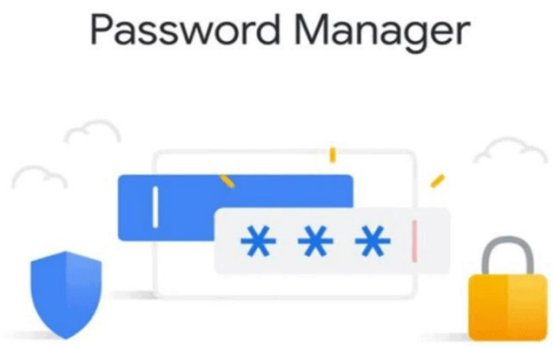
Improve Your Online Security - Password Manager

WRITTEN BY LOK WING LAVIN WONG



This tutorial will discuss what a password manager is, reasons to use it, and explain how it works. It includes some Python code I have made as a sample to demonstrate how it simply operates with using terminal. With easy-to-follow instructions and the essential screenshots, such as the coding and working processes, it will be more valuable to read. Whether you are a busy executive at a corporation or a regular citizen, you undoubtedly have a number of internet accounts today for a variety of purposes, such as email, social media, bank accounts, and so on. To protect these accounts containing sensitive data, strong passwords must be used. Use of a password manager is one effective safeguard.

Here is an overview of what is covered:

1. What is a Password Manager? Examples?

A password manager is a tool that can help you save passwords by securely storing, managing, and encrypting them (Bonneau, Herley, Oorschot, and Stajano., 2012). It is basically a database that stores a user's passwords and login information for the last several websites (Li, 2014). Instead of having to remember multiple, complex passwords

for different websites and software, only one master password is needed to access your password manager. You may already be familiar with or have tried a few passwords manager software, such as Dashlane, 1Password, KeePassXC, and others.

2. How does a password manager function?

A master key or master password is a unique and complex password that the user creates. Users only need to remember their master key, saving them from having to remember many log-ins and passwords. However, password managers are not limited to household passwords. They can also save safe notes, credit card details, and other vital private data. Some also include security features that alert you when a password has been hacked and request that you change any overused or shaky passwords (Gallagher, 2019). Some password managers can create strong, unique passwords for your internet accounts and store them in an encoded format (Liu, Y.-T. et al., 2018).

3. Reasons to use a password manager

This is done for convenience and security reasons. Due to the typical user's multiple web accounts and Internet-connected devices, users frequently use a password manager to

keep track of all of their passwords. According to research, the primary factor that could persuade people to utilise a password manager is that it has features that users find to be the easiest to use (Mayer, Munyendo, Mazurek, and Aviv., 2022). Additionally, modern password managers always come with extra security features like two-factor authentication or a key code for further access. A password manager thus offers us advantages in terms of ease and security.

4. Password Manager in Python (self-work)

Using different programming languages, such as C, C++, or Python, anyone can develop a password manager. The code used to create my own password manager using Python will be demonstrated. The following describes how it operates as an example:

Using terminal to apply Python code (command prompt)

The keyboard's "Enter" button is going to be utilised after entering any numbers or characters.

- Master Password, Main Menu and Create new password files

```
def main():

    clear()

    #initialize variables
    password = {}   # a dictionary to store the passwords loaded from the key files
    pm = PasswordManager()

    #check/create master password
    if pm.masterPW_is_created() == False:
        pm.create_key("key")
        var = input("Create a master password: ")
        pm.create_masterPassword(var)
    else:
        pm.load_key("key")
        str1 = input("Enter your master password: ")
        while str1 != pm.load_masterPW():
            str1 = input("Wrong master password, try again: ")

    done = False

    #menu choice
    while not done:
        # print(pm.key)
        # print(pm.password_dict)
        # print(pm.password_file)
        clear()
        pm.print_main_menu()
        choice = input("Enter your choice: ")
        if choice == "1":
            clear()
            path = input("Create a file name: ")
            pm.create_password_file(path+".txt")
        elif choice == "2":
            clear()
            pm.manage_pwFile_menu()
        elif choice == "q":
            done = True
            print("Bye")
        else:
            print("Invalid choice!")

if __name__ == "__main__":
    main()
```

```
def print_main_menu(self):
    print("""What do you want to do?
    (1) Create new password file
    (2) Manage existing password file
    (q) Quit
    """)
```

A master password (MP) is necessary as the first step to begin utilising the password manager; otherwise, you will not be able to access the Main Menu screen. First things first: when creating an MP with characters or numbers, keep in mind that this will serve as the passcode for the password manager, and you must use it to gain access in the future.

```
Create a master password: master
```

You will be taken to the password manager's main menu after creating the MP and pressing "Enter."

```
What do you want to do?
        (1) Create new password file
        (2) Manage existing password file
        (q) Quit

Enter your choice:
```

From the main menu, you can create a new password file or manage an existing password file. In order to create a new file for a first-time user, they must enter the number "1" followed by enter. To access the file's contents, you must give it a name, such as "university," and a password, such as "123." There can be as many files as you need. Once all of your passwords have been saved, select "Quit" to end the password manager.

```
What do you want to do?
        (1) Create new password file
        (2) Manage existing password file
        (q) Quit

Enter your choice: 1
```

```
Create a file name: university
Create a password for this file: 123
```

- **Manage password files (containing password generator)**

```
def manage_pwFile_menu(self):

    # print all password files
    txtfiles = []
    for file in glob.glob("*.txt"):
        txtfiles.append(file)
    print("Password files you have: ")
    print("\n".join(txtfiles))

    # enter the file to be loaded and prompt the user to input a password for this file
    path = input("Choose a file you want to manage (enter q to exit): ")
    if path == 'q':
        return
    if self.load_password_file(path) == -1:
        return

    file_password = input("Enter the password of this file: ")

    while file_password != self.password_dict["file password"]:
        file_password = input("Wrong password! Try again (input q to exit): ")
        if file_password == "q":
            return

    # print menu when password is correct
    done = False

    while done is not True:
        clear()
        print("""What do you want to do?
        (1) Add a password
        (2) View this password file
        (q) Quit
        """)
        choice = input("Enter your choice: ")
        if choice == "1":
            self.add_password()
        elif choice == "2":
            print("\n")
            for key, value in self.password_dict.items():
                if key == "file password":
                    continue
                print(key, ' : ', value)
            print("\n")
            input("Click enter to continue")
        elif choice == "q":
            done = True
        else:
            print("Invalid choice!")
```

You will be routed to the main menu when the creation of new files has been successful. Type "2" followed by the number to enter the next screen. You can now see the file name that has been set with ".txt" as the file format. In addition to the file name, such as "university.txt," you must input one of the file names exactly as they are stated. Undoubtedly, the password must be applied along with the file's access key.

```
What do you want to do?
        (1) Create new password file
        (2) Manage existing password file
        (q) Quit

Enter your choice: 2
```

```
Password files you have:
business.txt
university.txt
Choose a file you want to manage (enter q to exit): university.txt
Enter the password of this file: 123
```

To store your password, to type in ‘1’ and it will ask for a site name (e.g. LinkedIn), same as asking ‘where is the following password used’. Next, put down your own password with choosing ‘1) Enter your password’, else ‘2) Generate a password’ which is a password generator giving out a random password consists of characters and numbers.

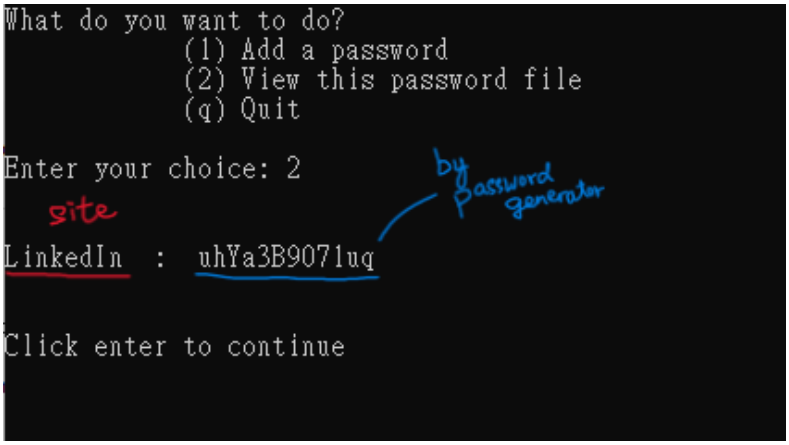
```
What do you want to do?
        (1) Add a password
        (2) View this password file
        (q) Quit

Enter your choice: 1
Enter the site: LinkedIn

        1) Enter your password
        2) Generate a password
        q) Quit

Enter your choice: 2
```

By entering "1" when prompted for a site name (such as LinkedIn), you can store your password and keep track of where the next password is used. The next step is to enter your own password by selecting "Enter your password," or you can use the password generator to generate a random password made up of letters, numbers, and symbols by selecting "Generate a password."



By applying the main menu, create files, and manage files functions, the entirety of the Python code has been displayed in the terminal.

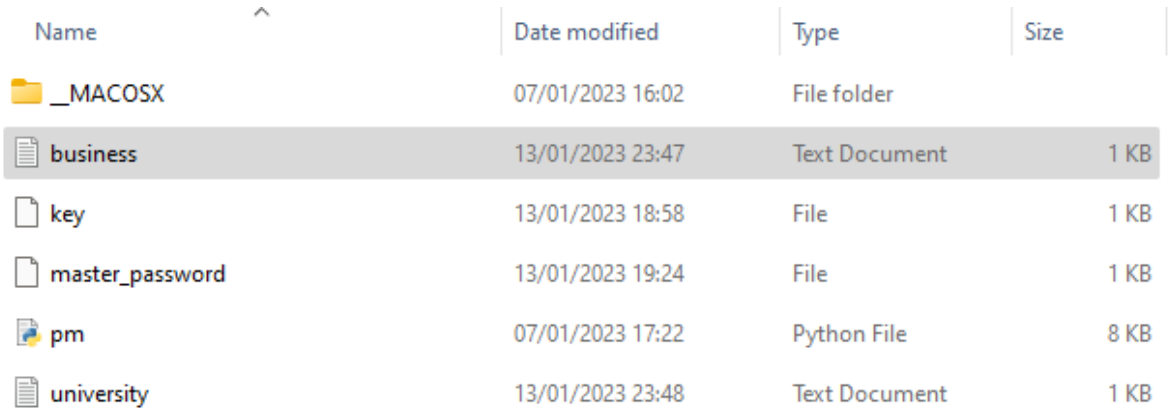
5. Differences between a well-known password manager and a self-made one

In addition to using a self-made password manager, consumers frequently use paid-for in-app purchases or free software to safeguard their passwords, such as KeePassXC, a well-known open-source password manager. Contrast the self-coding password manager as described with an example of a well-known password manager, such as KeePassXC. There are some significant variations between the two.

- **Control over code and functionality**

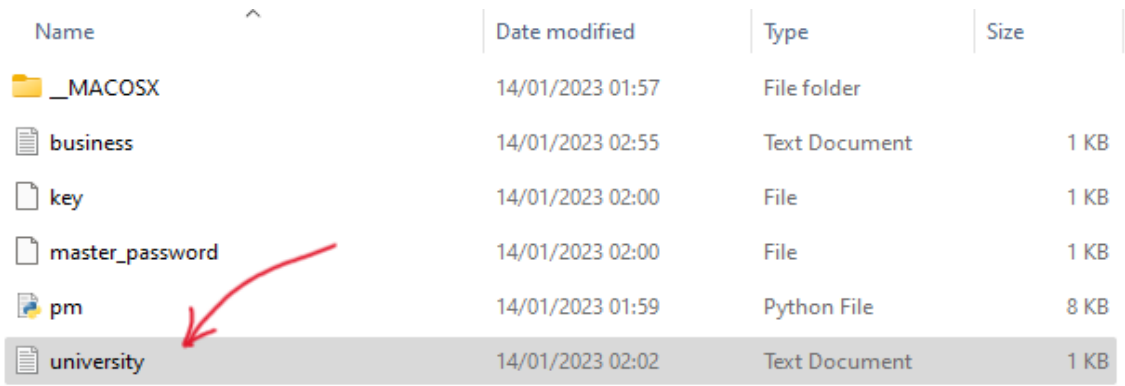
One of the key benefits of the user-created Python password manager is total control over the code and functionality. This implies that the password manager can be modified to meet the user's particular requirements. Additionally, the user has complete control over data storage and security, which is something that certain users are constantly concerned about. Additionally, it is less expensive to produce and maintain for business, non-profit, or individual use. An organisation can have better control over their passwords and secure their systems by using a self-made password manager. In reality, having a customised solution that meets the unique requirements of the organisation lowers the possibility of a future attack.
Cryptography with Fernet is used in the above self-made password manager for code encryption and decryption. The Cryptography package in Python offers low-level interfaces as well as high-level recipes using standard cryptography techniques,

including symmetric ciphers, message digests, and key derivation functions. Fernet is a sample with a high-level interface (Bray, S.W., 2020). In truth, the self-made password manager creates independent password files in text document format in the password manager's directory file after creating a file each time.

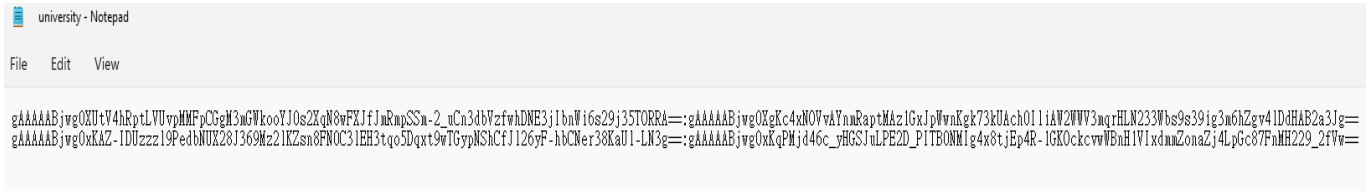


Name	Date modified	Type	Size
_MACOSX	07/01/2023 16:02	File folder	
business	13/01/2023 23:47	Text Document	1 KB
key	13/01/2023 18:58	File	1 KB
master_password	13/01/2023 19:24	File	1 KB
pm	07/01/2023 17:22	Python File	8 KB
university	13/01/2023 23:48	Text Document	1 KB

You will notice incredibly complex characters, numbers, and symbols when you open the university file. This is what Fernet approaches achieve when encrypting user input data. With the help of the "fernet encryption" symmetric encryption method, the data is protected from being read or changed without the key. Fernet encryption uses public-key cryptography standard 7 (PKCS7) padding, hash-based message authentication code (HMAC) using SHA-256 for authentication, and 128-bit advanced encryption standard (AES) symmetric encryption in cypher block chaining (CBC) mode. The secret key as ciphertext, encoding using the encrypted technique, and decoding using the decrypted function are all built-in functions of the cryptography package's Fernet module. (Ali, G. and Sam, A. E., 2021). Due to the numerous security methods that it applies, the Fernet encryption has been selected. Higher levels of security, reliability, privacy, authentication, authorization, data integrity, and secrecy are provided by Fernet. It guarantees non-repudiation and guards against brute-force attacks, identity theft, impersonation, shoulder-surfing, and phishing threats.



Name	Date modified	Type	Size
_MACOSX	14/01/2023 01:57	File folder	
business	14/01/2023 02:55	Text Document	1 KB
key	14/01/2023 02:00	File	1 KB
master_password	14/01/2023 02:00	File	1 KB
pm	14/01/2023 01:59	Python File	8 KB
university	14/01/2023 02:02	Text Document	1 KB



- **Open-source and free**

However, KeePassXC has a number of benefits over the Python password manager that was written. It can efficiently assist in securing a user's online presence and is first and foremost free. Since the programme is open-source, users can view the source code and modify it as they think appropriate. Everyone is welcome to offer ideas or improvements to improve the software, and many technological professionals have had the opportunity to analyse the code to evaluate its functionality and security (Master, A. 2022). Moreover, KeePassXC is also cross-platform, available for Windows, Linux and macOS, and can be run directly from a USB drive, providing an added layer of security as well. For details of KeePassXC, you access the link of reference 8.

- **Limited security and encryption**

KeePassXC's level of security and encryption may be superior to that of a Python password manager that was created. The performance, scalability, technical support, and customer service all could be constrained. The User Interface of KeePassXC, on the other hand, may be less user-friendly for some users and offer less customisation choices. User needs and tastes in particular will ultimately determine which option they choose. Using one or both of these techniques can improve an organisation's overall security and lessen the likelihood of further attacks.

6. Summary

In this tutorial, we covered some background information and fundamentals about password managers, demonstrated how a Python-coded password manager functions, and then compared it with an alternative tool, KeePassXC. We also discussed about some benefits of using a password manager to help a company improve its security.

Copy of full code

```python
from cryptography.fernet import Fernet
import os
import glob
import random
import string

# a function to clear the terminal screen
def clear():

    # detect operating system
    if os.name == 'nt': #for windows
        _ = os.system('cls')

    else:               #for mac and linux
        _ = os.system('clear')

# a function to generate a random string
def generate_random_string(len):
    return ''.join([random.choice(string.ascii_letters + string.digits) for n in range(12)])

class PasswordManager:

    def __init__(self):
        self.key = None
        self.password_file = None
        self.password_dict = {}

    def print_main_menu(self):
        print("""What do you want to do?
        (1) Create new password file
        (2) Manage existing password file
        (q) Quit
        """)

    def manage_pwFile_menu(self):

        # print all password files
        txtfiles = []
        for file in glob.glob("*.txt"):
            txtfiles.append(file)
        print("Password files you have: ")
        print("\n".join(txtfiles))

        # enter the file to be loaded and prompt the user to input a password for this file
        path = input("Choose a file you want to manage (enter q to exit): ")
        if path == 'q':
            return
        if self.load_password_file(path) == -1:
            return

        file_password = input("Enter the password of this file: ")

        while file_password != self.password_dict["file password"]:
            file_password = input("Wrong password! Try again (input q to exit): ")
            if file_password == "q":
                return

```

```
58          # print menu when password is correct
59          done = False
60
61          while done is not True:
62              clear()
63              print("""What do you want to do?
64              (1) Add a password
65              (2) View this password file
66              (q) Quit
67              """)
68              choice = input("Enter your choice: ")
69              if choice == "1":
70                  self.add_password()
71              elif choice == "2":
72                  print("\n")
73                  for key, value in self.password_dict.items():
74                      if key == "file password":
75                          continue
76                      print(key, ' : ', value)
77                  print("\n")
78                  input("Click enter to continue")
79              elif choice == "q":
80                  done = True
81              else:
82                  print("Invalid choice!")
83
84      def masterPW_is_created(self):
85          return os.path.exists("master_password")
86
87      def create_key(self, path):
88          self.key = Fernet.generate_key()
89          with open(path, 'wb') as f:
90              f.write(self.key)
91
92      def load_key(self, path):
93          with open(path, 'rb') as f:
94              self.key = f.read()
95
96      def create_masterPassword(self, master_password):
97          with open("master_password", 'a+') as f:
98              f.write(Fernet(self.key).encrypt(master_password.encode()).decode())
99
100     def load_masterPW(self):
101         with open("master_password", 'rb') as f:
102             encrypted_masterPW = f.read()
103             decrypted_masterPW = Fernet(self.key).decrypt(encrypted_masterPW)
104             return decrypted_masterPW.decode()
105
106     def create_password_file(self, path, initial_values=None):
107         self.password_file = path
108         # check if file exist
109         if os.path.exists(path):
110             input("Password file exist! Press enter to continue")
111             return
112
```

```
113         # prompt user to create a file password
114         file_password = input("Create a password for this file: ")
115 
116         with open(path, 'w') as f:
117             encrypted_site = Fernet(self.key).encrypt("file password".encode())
118             encrypted_password = Fernet(self.key).encrypt(file_password.encode())
119             f.write(encrypted_site.decode() + ":" + encrypted_password.decode() + "\n")
120         if initial_values is not None:
121             for key, value in initial_values.items():
122                 self.add_password(key, value)
123 
124     def load_password_file(self, path):
125         if os.path.exists(path) == False:
126             input("Password file not exist. Press enter to continue. ")
127             return -1
128         self.password_file = path
129 
130         #Decrypt every line in the loaded password file, then store it to a dictionary variable
131         with open(path, 'r') as f:
132             for line in f:
133                 encrypted_site, encrypted_password = line.split(":")
134                 decrypted_site = Fernet(self.key).decrypt(encrypted_site.encode().decode()).decode()
135                 decrypted_password = Fernet(self.key).decrypt(encrypted_password.encode().decode()).decode()
136                 self.password_dict[decrypted_site] = decrypted_password
137 
138     def add_password(self):
139         #Error checking: check if a password file is load
140         if self.password_file is None:
141             print("Load a password file first.")
142             return
143 
144         #prompt user to input the site and password they want to add
145         site = input("Enter the site: ")
146 
147         print("""
148         1) Enter your password
149         2) Generate a password
150         q) Quit
151         """)
152 
153         done = False
154         while done is False:
155             choice = input("Enter your choice: ")
156             if choice == "1":
157                 password = input("Enter the password: ")
158                 done = True
159             elif choice == "2":
160                 password = generate_random_string(12)
161                 done = True
162             elif choice == "q":
163                 return
164             else:
165                 print("Invalid choice!")
166 
167         self.password_dict[site] = password
```

```
168
169         #add pw to the loaded file
170         with open(self.password_file, 'a+') as f:
171             encrypted_site = Fernet(self.key).encrypt(site.encode())
172             encrypted_password = Fernet(self.key).encrypt(password.encode())
173             f.write(encrypted_site.decode() + ":" + encrypted_password.decode() + "\n")
174
175     def get_password(self, site):
176         return self.password_dict[site]
177
178 def main():
179
180     clear()
181
182     #initialize variables
183     password = {}   # a dictionary to store the passwords loaded from the key files
184     pm = PasswordManager()
185
186     #check/create master password
187     if pm.masterPW_is_created() == False:
188         pm.create_key("key")
189         var = input("Create a master password: ")
190         pm.create_masterPassword(var)
191     else:
192         pm.load_key("key")
193         str1 = input("Enter your master password: ")
194         while str1 != pm.load_masterPW():
195             str1 = input("Wrong master password, try again: ")
196
197     done = False
198
199     #menu choice
200     while not done:
201         # print(pm.key)
202         # print(pm.password_dict)
203         # print(pm.password_file)
204         clear()
205         pm.print_main_menu()
206         choice = input("Enter your choice: ")
207         if choice == "1":
208             clear()
209             path = input("Create a file name: ")
210             pm.create_password_file(path+".txt")
211         elif choice == "2":
212             clear()
213             pm.manage_pwFile_menu()
214         elif choice == "q":
215             done = True
216             print("Bye")
217         else:
218             print("Invalid choice!")
219
220 if __name__ == "__main__":
221     main()
222
```

Reference

1. J. Bonneau, C. Herley, P. C. v. Oorschot and F. Stajano, "The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes," 2012 IEEE Symposium on Security and Privacy, 2012, pp. 553-567, doi: 10.1109/SP.2012.44.

2. Liu, Y.-T. et al. (2018) “Splitpass: A Mutually Distrusting Two-Party Password Manager,” Journal of Computer Science and Technology, 33(1), pp. 98–115. doi: 10.1007/s11390-018-1810-y.

3. Gallagher, E. A. (2019) “Choosing the Right Password Manager,” Serials Review, 45(1), pp. 84–87. doi: 10.1080/00987913.2019.1611310.

4. Li, Z. H. (2014). The Emperor’s New Password Manager: Security. CALIFORNIA UNIV BERKELEY DEPT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCES. Retrieved 07 07, 2014, from http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-138.html

5. Mayer, P., Munyendo, C.W., Mazurek, M.L. and Aviv, A.J., 2022. Why Users (Don't) Use Password Managers at a Large Educational Institution. In 31st USENIX Security Symposium (USENIX Security 22) (pp. 1849-1866).

6. Bray, S.W., 2020. *Implementing Cryptography Using Python*. John Wiley & Sons.

7. Ali, G. and Sam, A. E. (2021) “A Secure and Efficient Multi-Factor Authentication Algorithm for Mobile Money Applications,” Future Internet, 13(12), pp. 299–299. doi: 10.3390/fi13120299.

8. Master, A. (2022). PASSWORD MANAGERS: SECURE PASSWORDS THE EASY WAY. The Center for Education and Research in Information Assurance and Security of Purdue University, from https://ag.purdue.edu/department/asec/ext-pub-ace15w.html

TASK	POOR 0-39%	ACCEPTABLE 40-49%	GOOD 50-59%	VERY GOOD 60-69%	EXCELLENT 70-84%	OUTSTANDING 85%+
REPORT STRUCTURE AND REFERENCES (10%)	Lacking logical structure. Missing terminology from the field, or incorrect application of terminology.	Lacking structure, each sentence appears fragmented and unrelated points follow each other awkwardly. Language/style not appropriate for a technical audience (assumes too much knowledge, condescending).	Structured well, with terminology from the field applied correctly. Language/style appropriate for an expert in the area, but not someone new to this area (e.g. Nmap for a penetration tester) An attempt at references but these are just URLs pasted into the document.	Use of headings and subheadings, all points present in some form. Language style at the appropriate level for a technical individual new to this area of security. Some references.	All points present with correct formatting, an attempt to follow Harvard or IEEE references. Language style appropriate for an executive audience in a professional blog. Some academic references.	Correct formatting, Harvard or IEEE references and each figure is labelled correctly Language style appropriate for an executive audience in a professional blog and can be published without any stylistic changes. Proper academic sources and RFC's used.
TOOL CHOICE & CODE (20%)	No tool is discussed or presented. Screenshots presented are copied from the lab worksheets, rather than your own work.	Some tooling is discussed, but it is not clear which exact tool.	Tool was covered in class or is one of the examples from the assessment.	Tool was covered in class or is one of the examples, with some mention of workflow between tools, e.g. Nmap to Metasploit	Tool was not covered in class or was developed for the assessment in Bash or Python.	Tool was developed for the assessment in Bash or Python, with accompanying code.
TUTORIAL (30%)	Tutorial missing.	Limited tutorial, demonstrating a non-primary feature. Some link to further reading.	Very good tutorial which demonstrates the tools primary feature Clear link to further reading, with some scaffolding "to learn more about x features you can read y article" Limited case for 'why' this tool.	Clear tutorial which presents the core feature of tool with some discussion of alternative features. Link to further reading for alternative features. A strong 'why' case is presented for the tool	Clear tutorial presenting multiple features of the tool. Link to further reading for each feature linking to another tutorial, book, or documentation Tutorial discusses alternative tools to achieve the same result	Very in-depth tutorial, demonstrating multiple features of the tool in great technical detail. References to further reading to specific documentation providing Tutorial critically analyses alternative tools.

MITIGATION (10%)	No discussion on reducing attack risk with this tool.	Limited discussion to how using this tool contributes to security overall at an organisation.	Good discussion on how this tool contributes to security.	Clear discussion on how using this tool can contribute to overall security at an organisation and reduce the opportunity for another attack.	Very good discussion on the mitigation potential of this tool within the context of an organisation.	Very good discussion on the mitigation potential of this tool, including links to historical or theoretical attacks.
SCREENSHOTS (10%)	No screenshots in the tutorial.	Screenshots may be too small to read but are present, or the screenshots are cut making them difficult to read	Screenshots can be read easily and at a reasonable zoom level.	Screenshots have been cropped and are easily read at a reasonable zoom level, with relevant information highlighted and/or visible.	Screenshots are clear, highlight only the important information and have appropriate labels under each.	Screenshots are labelled and are referred to in the text with 'Figure 1', 'Figure 2'
SELF REFLECTION (20%)	No annotated copy of the marking grid	An annotated copy of the marking grid but with some annotations missing.	Annotated copy of the marking grid without any supporting comments.	A copy of the marking grid with some supporting comments but unorganised explaining how they have met that grade boundary.	A copy of the marking grid with a bullet point for each annotation, easy to read and organised. Evidence of an honest self-reflection.	An annotated copy with supporting comments and a link with evidence (paragraph 2, sentence 2).

REPORT STRUCTURE AND REFERENCES

- Had use of 8 academic references as citation with Harvard reference method, last page is references' information (Contents of heading 1, 2, 3, 5)
- Tried to use an objective language style, not to be subjective, but not sure if the tone is appropriate to be a professional blog post or not

TOOL CHOICE & CODE

- Choose password manager as a security tool
- Presented the workflow of my coded password manager with steps and screenshot explanation (Content of heading 4, page 3-6)
- Code developed in Python
- Full copy of code (page 10-13)

TUTORIAL

- Reckon that could be the clearest presentation of the tutorial (Content distribution)
- Did demonstrate the primary feature, create a password file for store password details (Heading 4, Manage password files (contains password generator), Page 5-6)
- Explain reasons to use password manager (Heading 3 & 5; Page 3, 9)
- Stated an alternative tool – KeePassXC (Subheading "Open-source and free", Page 9)
- Given direction for further reading (Subheading "Open-source and free", last sentence, Page 9)

MITIGATION

- Reckon that had a clear discussion on using password manager for security (Subheading "Control over code and functionality", Page 7-8)
- Pointed out using password manager to contribute an organisation, but it should be more (Content of heading 5, Page 8-9)

SCREENSHOTS

- Attached screenshots with every step of workflow with my code (Full content of Heading 4, Page 3-7)
- Screenshots of password files and password and details encoded (Subheading "Control over code and functionality", Page 8-9)
- Screenshots of full code of my password manager (Page 10-13)
- Did cropped screenshot to be easily read with relevant information and labels (Page 3-9)

SELF REFLECTION

- Did a copy of the marking grid with bullet points for each part of grade boundary highlighted.
- Separated each part of task to be clearly and easily read
- Provided pages and where, the regarding written points can be found

Things to improve / things done well:

- Reckon that I found the topic which contains fewer academic references to support my points
- Should make a mind map of different ideas next time and select a topic with contains more references
- Never tried to write code in Python and finally succeed with my full code which is workable
- Not sure if I wrote the blog post in a correct structure, hope to know more and write another one better in the future
