THE PASSWORDLESS MANAGER

Țurcan Nicolae
Master in Economics
Computer Security University of Oklahoma, Norman, USA
e-mail: nicolae@ou.edu

Schneider Tobias
Master in Economics
Computer Security University of Oklahoma, Norman, USA
e-mail: tobias.schneider-1@ou.edu

Abstract
Password managers are common software solutions to store secure and valuable information like banking credentials, passwords or personal information. Our goal is to design a password manager using a new approach to manage private data. We analyze problems of existing solutions, propose our own design and technique, and finally implement a solution. Our application does not store passwords in any form on the memory but instead generates them on the fly by processing the user input and other variables combined with a key derivation function. Finally, we elaborate limitations of our approach and suggest future work.

Keywords: *security, managers, password, system design, security analysis*

JEL Classification: *C63*

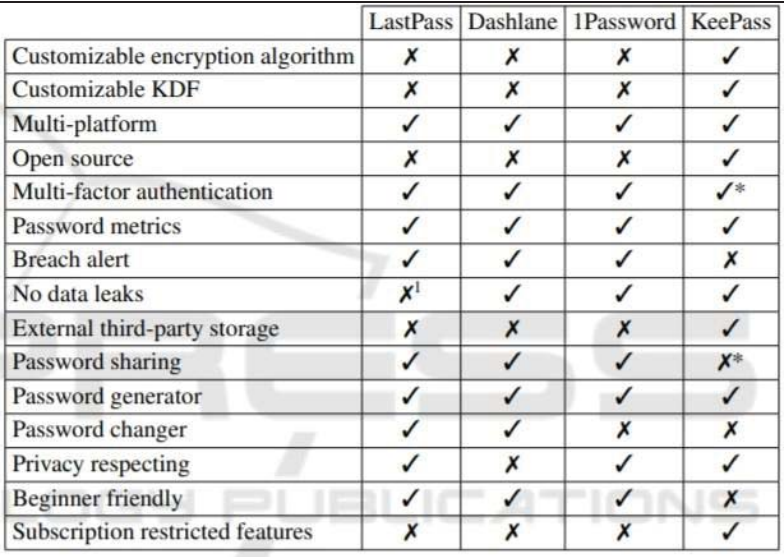
INTRODUCTION

A password manager is a computer program which stores and administrates secret data like passwords on a computer, smartphone or in web applications. These programs originate from the problem that users need secure passwords on their system and many websites. There is a high security risk of using the same usernames and passwords for different services, as a single stolen password would allow access to all services. Therefore, many different passwords are needed. To protect these accounts from unauthorized access, many and long passwords are needed. A possible solution to tackle this problem is using a password manager to store the valuable information in an encrypted format.

Some password manager utilize a master key to encrypt the private data. The security of a master password approach depends on the strength of the mostly user chosen passphrase. If an attacker is able to obtain the master key all saved passwords are rendered vulnerable. The password database, also called vault is stored on the file system and has to be encrypted. Also, the used encryption algorithm must be implemented flawlessly to avoid security problems which are risk to exploitation by hackers.

In 2014, more than five million Gmail logins were leaked. A password manager company named LastPass conducted an analysis of that data and came up with the following findings [8]:

- Top 10 most-used passwords were 123456, password, 123456789, 12345, qwerty, 12345678, 111111, abc123, 123123, 1234567.
- Top 10 most-used words in passwords were password, qwerty, love, monkey, dragon, hello, iloveyou, abcd, welcome, July.



	LastPass	Dashlane	1Password	KeePass
Customizable encryption algorithm	✗	✗	✗	✓
Customizable KDF	✗	✗	✗	✓
Multi-platform	✓	✓	✓	✓
Open source	✗	✗	✗	✓
Multi-factor authentication	✓	✓	✓	✓*
Password metrics	✓	✓	✓	✓
Breach alert	✓	✓	✓	✗
No data leaks	✗[1]	✓	✓	✓
External third-party storage	✗	✗	✗	✓
Password sharing	✓	✓	✓	✗*
Password generator	✓	✓	✓	✓
Password changer	✓	✓	✗	✗
Privacy respecting	✓	✗	✓	✓
Beginner friendly	✓	✓	✓	✗
Subscription restricted features	✗	✗	✗	✓

Figure 1. Overview of popular password managers

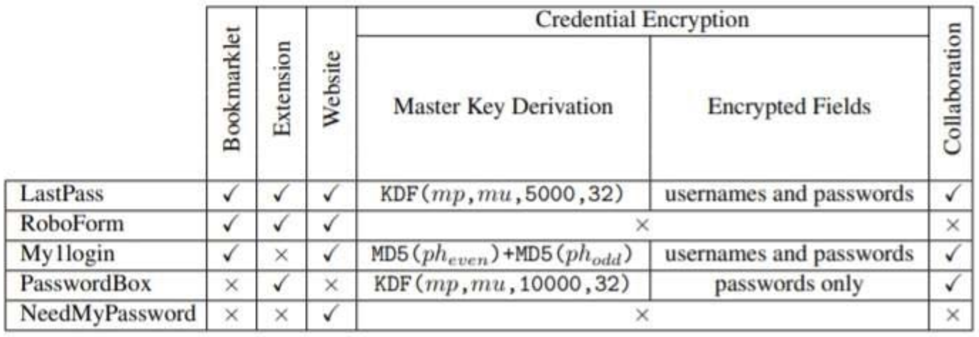
– 92.96% of the passwords were 1-10 characters in length The above statistics suggest that average users are not aware of how insecure their password choices or do not realize how many other people tend to use the same combination. This piece of knowledge makes the job of hacking a particular password much easier for those who are interested of stealing such personal information. Therefore, a secure master key is essential for the security of the saved passwords.

Instead of storing the encrypted passwords on the file system we propose a new approach which avoids the use of a vault. We generate passwords on the fly by calculating a function which takes a combination of user name, website, master key and a counter as input. Chapter II presents related work to our research. Chapter III explains our system, used algorithms and password generation in detail. In chapter IV we propose our application by explaining the user interface and implementation details. Also, chapter V contains a security analysis of our approach and possible attacks. Chapter VI summarizes the results of our research.

RELATED WORK

We decided to explore the most popular password managers in order to come up with a hypothesis as to why people do not tend to use password managers more often. The paper presenting SAFEPASS, which is a password manager, has summed up the main features of four of the most popular password managers. [1]

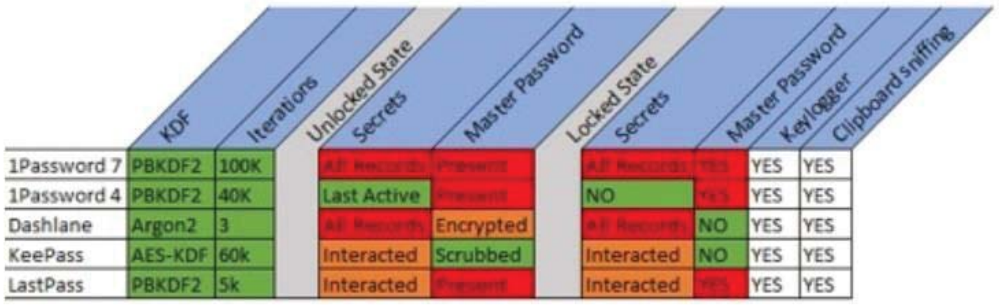
As it can be seen in the table above, only KeePass provides customizability regarding the encryption algorithm used for storing a password. This feature might be of interest for people with encryption knowledge and who desire to take control over how their passwords can be encrypted, perhaps mixing the algorithms for different instances. On the other hand, KeePass is the manager that is the least beginner friendly when compared to LastPass, Dashlane, and 1Password.



	Bookmarklet	Extension	Website	Credential Encryption		Collaboration
				Master Key Derivation	Encrypted Fields	
LastPass	✓	✓	✓	KDF(mp,mu,5000,32)	usernames and passwords	✓
RoboForm	✓	✓	✓	×		×
My1login	✓	×	✓	MD5(ph_{even})+MD5(ph_{odd})	usernames and passwords	✓
PasswordBox	×	✓	×	KDF(mp,mu,10000,32)	passwords only	✓
NeedMyPassword	×	×	✓	×		×

mu: master username mp: master password
ph: passphrase $ph_{even(odd)}$: characters at even (odd) positions of ph
KDF(p,s,c,l) is a key derivation function [23], which derives key of length l octets for the password p, the salt s, and the iteration count c.

Figure 2. Encryption schemes of popular password managers



	KDF	Iterations	Unlocked State Secrets	Unlocked State Master Password	Locked State Secrets	Locked State Master Password	Keylogger	Clipboard sniffing
1Password 7	PBKDF2	100K	All Records	Present	All Records	YES	YES	YES
1Password 4	PBKDF2	40K	Last Active	Present	NO	YES	YES	YES
Dashlane	Argon2	3	All Records	Encrypted	All Records	NO	YES	YES
KeePass	AES-KDF	60k	Interacted	Scrubbed	Interacted	NO	YES	YES
LastPass	PBKDF2	5k	Interacted	Present	Interacted	YES	YES	YES

Figure 3. Summary of password managers security items

A study from 2014 analyzed five popular password man- agers: LastPass, RoboForm, MyLogin, PasswordBox, and NeedMyPassword. It concluded that all five managers had critical vulnerabilities, and with four of them arbitrary user credentials could be stolen [2].

The study also provided guidance and mitigations to avoid the vulnerabilities. The goal of our password manager is to follow this guidance in order to avoid critical vulnerabilities.

Additionally, that study presented the following table that provides a summary of certain features for those five password managers.

The above table shows that not all password managers combine the bookmarklet, extension, and website format for user interaction. We choose to keep our password manager as simple as possible to avoid misuse and prevent security issues. In 2019, Independent Security Evaluators (ISE) have con- ducted an analysis four major password managers: 1Password, Dashlane, KeePass, and LastPass. The main focus of the anal- ysis was to explore the sanitization of sensitive information stored by the password managers [3]. The following figure summarizes their findings.

The red in the figure above displays security violations of proposed security guidelines by ISE. The conclusion of the study was that each password manager analyzed failed to implement proper sensitive information sanitization for various reasons. The most urgent concern outlined was when a password manager was put into a locked state and sensitive information was not sanitized.

More work related to password managers has been done in the past. For instance, a solution that created a password manager for the Firefox browser using cloud-based storage was outlined in 2014 [4]. Since this solution does not target Chrome as its primary browser, the fact that our password manager will primarily be developed for Chrome will setus apart. Two more cloud-based password solutions were outlined in the papers that presented Passpet [5], and Password Multiplier [6].

Passpet and Password Multiplier represent for the most part solutions that solely help in generating passwords, hence our password management capabilities will offer a different approach to solve the password problems.

A password manager named Master Password is the most similar application compared to ours [11]. The main idea is the provision of passwords only when the user needs to. If a user selects a website the tool generates the password. This application still stores a version of the hashed master key on the file system to allow the user to login. Even if the passwords are erased after the generation the hash of the master key is still saved on the memory.

Based on the above observation, we decided to come up with a solution that will offer a combination of features that is not currently present on the market. In particular, our password manager will be secure, beginner friendly, appealing to people who require more customizability, secure according to ISE guidelines.

SYSTEM DESIGN

This chapter explains the main idea of our approach. The system design is fundamental for the security of the applica- tion. Criterias for the chosen encryption, password generation and functionality are explained in detail.

A. *System Overview*

As our aim is to not store any passwords in neither en- crypted nor unencrypted form on the memory so we propose a new take on a password manager. If the we do not utilize a password database we have to generate the passwords by processing information the user inputs and data which does not have to be necessarily private. This way we can store the name of the website on the computer but every other information is secret and supplied by the user to generate the password for the requested web page.

Figure 4 shows the concept of our system. The name of the user, the website and the counter are combined into a message string. This message is part of the input for the password generation function. The user input is displayed on the left and consists of three inputs variables:

- **Master key:** The passphrase which is used to gain access to the application. Usually, the hash of the master key is stored and compared with the hashed input of the user. As we do not store the key on the file system the passphrase has to be correct or a wrong password will be generated.
- **User name:** The name of the user does not have to be secret but the number of possible inputs increases if the user name remains unknown.
- **Website:** The name or URL of a web page. The list of websites is saved in the application and is not secret.

Additionally, we use a counter which increments every time a new password is generated. Otherwise would the input for the generation of a new password for the same website result in the same passphrase.

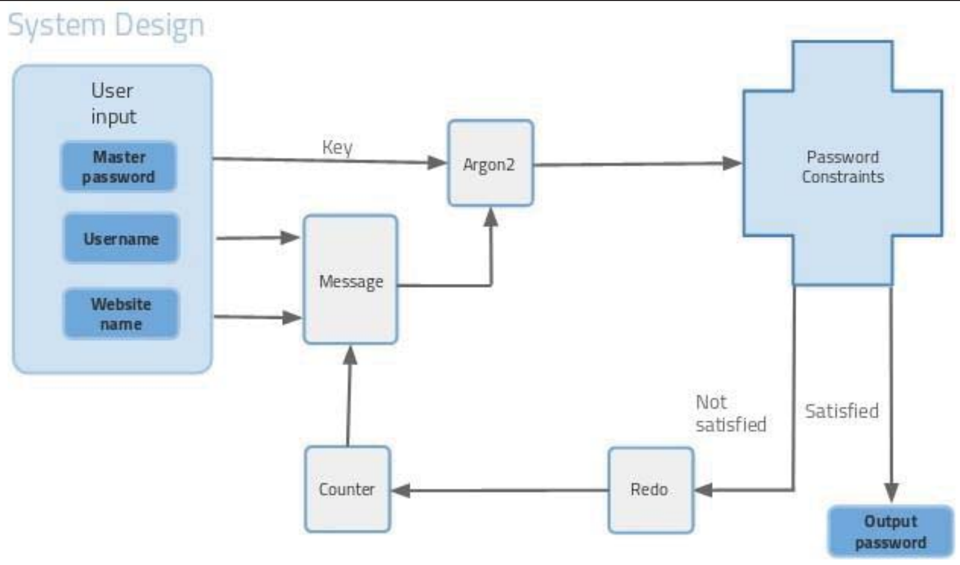


Figure 4. System Design

The function for the password generation must meet several requirements to be suitable for our use case. We use a hash function *H* for this purpose. The most important requirement is the one-way property. If an attacker is able to obtain the password *H*(*m*) it must be computationally infeasible to find the original message *m*.

We use a key derivation function (KDF) for the password generation. Such a function repeats the hash process many times to make password cracking more difficult. The KDF derives a key from a secret value which would be the com- bination of master password and message in our case. The advantage of such a function compared to a simple hash function is the slow down of the computation to aggravate brute-force and dictionary attacks. The number of iterations indicates how often the hash function is calculated. The result of the previous function is the input for the next calculation. The number of iterations should be high enough to slow down the calculation process to a reasonable time. The usual input for a key derivation function is the string to hash, a salt and the number of iterations for the sub-function. The use of a salt prevents an attacker from precomputing rainbow tables of a dictionary or random words.

To choose a key derivation function fitting our purposes we examine different available solutions. The PBKDF2 [https://www.ietf.org/rfc/rfc2898.txt] (Pass- word Based Key Derivation Function 2) is used for deriving a key from a password. It possible to specify the hash algorithm, length of the key and the number of iterations. A disadvantage is that it can be implemented in hardware with a small curcuit and little RAM very cheap. The HKDF (HMAC- based Extract-and-Expand Key Derivation Function) is used to derive keys of a fixed length for further crpytographic usage [https://tools.ietf.org/html/rfc5869].

Another KDF is argon2 [https://tools.ietf.org/html/draft-irtf-cfrg-argon2-03] which was selected as winner of the Password Hashing Competition in 2013. The algorithm creates a big vector in the memory to avoid the cracking by specialized hardware which usually has a small memory. We select argon2 as key derivation function due to its security features compared to the other algorithms.

We define some constraints for the generated password to meet the requirements requested by most modern websites:

– Minimum length is 8 characters
– Contain at least one uppercase letter
– Contain at least one number

– Contain at least one special character

Finally, if the derived password complies with the required quality it is shown to the user in the interface. This password is used for the website which was selected by the user before.

B. *Secure Master Key*

In 2006, a study of 49 undergraduates has shown that the majority had three or fewer user passwords and passwords were reused twice [7]. Over time, password reuse has only increased because students created more accounts but did not come up with more passwords. This study is particularly inter- esting because students are the ones to adopt new technology the quickest, but even they did not adopt the use of password managers at a broad scale.

RoboForm has performed a survey which polled 1,000 random people in the US and UK about password security practices [9]. The following are some of the results they came up with:

- ✓ Only 27% never have their browser remember their passwords
- ✓ 42% write their passwords down, 10% save them in a document, 23% always use the same password, 25% use personal information, 5% enter as numbers on mobile device, 37% use phrases consisting of numbers and letters, 8% use a password manager.
- ✓ 12% use more than 11 passwords, 29% use between 6 and 10 passwords, and 59% use up to 5 passwords.
- ✓ 63% have forgotten a password or had a password com- promised in their professional life.
- ✓ 53% have forgotten at least a password for work in the past month.

The fact that only 27% never have their browser remember passwords might sound odd to some, however, browsers do not provide secure spaces for storing such information. For instance, a paper from University of California that analyzed the security of password manager database formats concluded that the way Google Chrome stores its users' passwords does not provide neither secrecy nor integrity [10].

Only 8% use password managers, meaning that users are overall still reluctant to use such kind of tools for password storage. Password managers solve the problem that people need to remember different passwords for several accounts. The user has to remember one master key to access all data. In many cases, the master key consists of a 12 to 16 character long, hard to remember passphrase. Since the security of such a system depends on the strength of the master key it is of highest priority to choose a strong pssword.

A possible solution to tackle the difficulty of remembering such a master key is the use of the Diceware passphrase [http://world.std.com/~reinhold/diceware.html]. This approach makes use of a list containing 7776 words of different length. Our application allows the user to generate such a password for his master key. In detail, the roll of five dices results in one word. We make use of a passphrase of six Diceware words so basically we roll the dice 65 = 30 times. The resulting password is a combination of these words, for example *PelicanTrustingUnmovableGemAstronautRipcord*.

Another argument for the usability is the following scenario. If the user inputs a wrong master key or flips a character the system generates the password based upon the wrong input and the computed passphrase is useless. After the login attempt the user recognizes that his password is false and tries to find the error in his usual master key for example *Sr0 pR7m&3K!*. It is difficult to find the error in such strings compared to a Diceware phrase. The problem that the user is not notified while entering the false password remains in both cases.

The advantage of using a Diceware password is the con- catenation of simple words. Such a phrase compared to a random combination of letters, digits and special characters is easy to remember. To compare the security of both options we calculate the entropy of both approaches. The entropy is a measurement for the set of random information in a system. Define password *p*1 as a 12 character random sequence of letters, numbers and some special chars (86 characters in total) and *p*2 as a Diceware password of 6 words. Calculate the entropy *H* as follows:

$$H(p1) = log2(8612) = 77.1\ bit\ H(p2) = log2(77766) = 77.5\ bit$$

As the entropy of both passwords is almost the same we conclude that the strength of both passwords is equal. Therefore, we provide the possibility to random a master key using Dicewords.

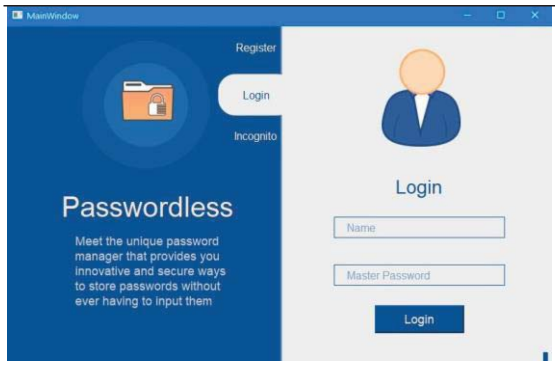
APPLICATION

This section presents the application and its workflow. Figure 5 shows the window that a first time user will be prompted with. The left hand side displays the application's name and a short summary of what it does. It also contains three navigation buttons: Register, Login, and Incognito. The right hand side presents a basic login form. In case the user does not have an account yet, he can create it using the register option on the left hand side of the application. After a successful registration, user's name and master password are hashed and stored in a local file for future login checks. In case the user does not want any of these to be stored anywhere, he can choose the Incognito option on the left hand side.

Figure 6 presents the window that logged in users are presented with. By default, logged in users are presented with the classic mode of our application. The left hand side offers information about the classic mode. The right hand side displays some social media presets that when clicked generate the according name in the website name section (i.e. if the Facebook logo is clicked, Facebook will be filled in the website name section). Otherwise, the user can manually type in the website for which he wants to generate a password. The blue label at the button will be the region where the generated password will appear.

Figure 7 is an example of how a generated password use case works. By either pressing the Instagram logo or manually typing it in, the user is presented with a generated password inside the blue label. This password has been generated by using the existed hashed of the user's name and password and combined with the website name provided. In case the user inputs the same website name in this section again, the same password will be generated. This way, the user can recover the already generated password without having to store it anywhere.

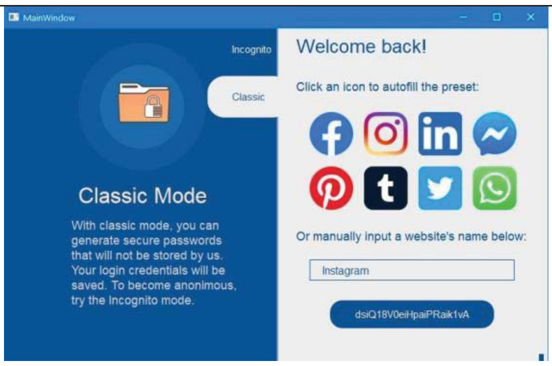
Figure 8 displays the incognito mode of Passwordless. The left hand side provides an explanation of the mode. The right hand side, just like in the classic mode, contains the social media logo shortcuts to fulfill the website name section. Incognito mode additionally requires the user to input the name and the master password because it does not use any stored information to generate the password.



Passwordless

Meet the unique password manager that provides you innovative and secure ways to store passwords without ever having to input them

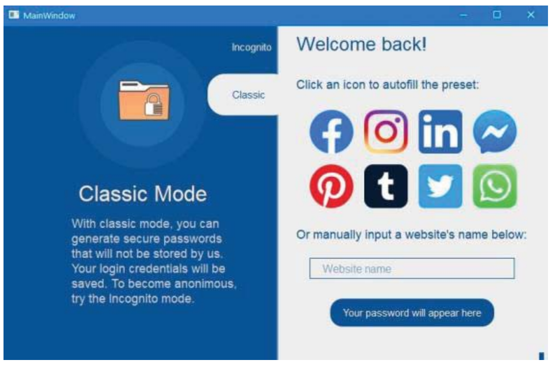
Figure 5. Default login page


MainWindow
Incognito
Classic
Welcome back!
Click an icon to autofill the preset:
Or manually input a website's name below:
Instagram
dsiQ18V0eiHpaiPRaik1vA

Classic Mode

With classic mode, you can generate secure passwords that will not be stored by us. Your login credentials will be saved. To become anonimous, try the Incognito mode.

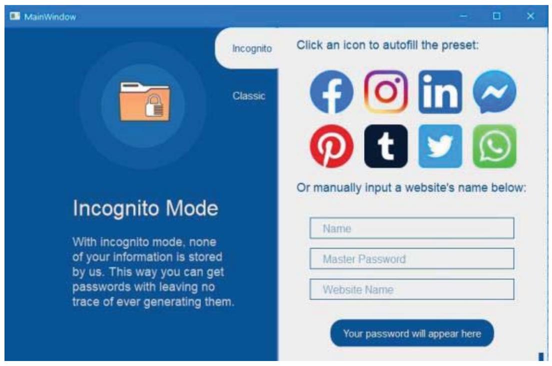
Figure 7. Classic Mode with generated password


MainWindow
Incognito
Classic
Welcome back!
Click an icon to autofill the preset:
Or manually input a website's name below:
Website name
Your password will appear here

Classic Mode

With classic mode, you can generate secure passwords that will not be stored by us. Your login credentials will be saved. To become anonimous, try the Incognito mode.

Figure 6. Classic mode


MainWindow
Incognito
Classic
Click an icon to autofill the preset:
Or manually input a website's name below:
Name
Master Password
Website Name
Your password will appear here

Incognito Mode

With incognito mode, none of your information is stored by us. This way you can get passwords with leaving no trace of ever generating them.

Figure 8. Incognito Mode

After typing in the name and the master password, while the user is inputting the website name the blue label will begin generating a password. As mentioned before, this mode calculates the passwords on the fly and does not use any stored data and does not store any of its results.

While developing the application, one of our goals was mak- ing sure that the application is easy to understand and usable. This is the reason why we decided to divide the interface into two pieces, the left piece explaining the current state of the application and the right piece focusing on user interaction. This way, the users do not have to search for specific button to understand what each application state entails. The usability was enhanced by using a simple minimalist design and limited interaction fields so that the user is not overwhelmed.

SECURITY ANALYSIS

The security of our system is a top priority target as flaws in the implementation result in risk of compromised user accounts. In this chapter we analyze the security of our application and propose challenges and limitations we currently deal with.

A. *Is our system design secure?*

It is necessary that the cryptographic operations in our system do not leak any information or are insecure in its execution. We assume that the input of the user is not intercepted by malicious software or other attacks. Since the hashing of the string combination of password, user name, website and counter is a safe operation the system itself is inherently secure. Security vulnerabilities in our application are more subject to errors in the implementation, memory leaks or attacks using video cameras.

The attacks on several password managers exploiting mem- ory leaks in 2019 revealed security issues affecting even pop- ular applications [3]. The authors were able to extract secrets from the memory during a locked state. Due to the compexity of this attack we did not implement any countermeasures since the attack is very complex, uses tools we do not have access to and requires more research into memory leakage on Windows systems.

Another way to recieve the master key is without using exploiting security vulnerabilities in the actual application. Spying on a users keyboard is a simple yet effective way to get the master key. This attack only requires a camera with a high resolution to detect the keystrokes.

Our system provides no security measures against malware or malicious software. Since we do not store the master password in any form it is not possible for any application to read out the key. Malware specialized on stealing store data is uneffective in our case.

B. *Challenges and Limitations*

As for every common password manager the user has to remember the master key. If the user forgets the passphrase it is impossible to access the passwords. We do not provide any recovery options as the master key is not saved on the file system. By providing the opportunity to make use of a Diceword password we aim to minimize the risk of forgetting the master key.

If a new user utilizes our application it is necessary to change the passwords for all existing accounts. If a new website is added to our list the system calculates a password for it. There is no way to simplify that step because our system generates passwords based upon the name of the website amongst other things. The time required increases the more accounts the user applys new passwords to. After this initialization step the password manager is ready for further usage.

Another limitation which includes other common password managers as well is the risk of compromising the master key. If the user notices critical activities on registered accounts it is necessary to change the password on every site which is stored in the application. This step is time-consuming but not avoidable by any security measure.

A further problem is the use of the application in an unsafe environment. If the computer is infected by a keylogger the master key can be obtained. Other common password manager face the same problem. This problem has to be tackled by the user of the computer and is not part of our research.

CONCLUSION

Finally, we were able to implement our approach that does not store passwords in any format on the filesystem. We use a key derivation function to generate passwords and propose a user friendly interface to complete our password manager. With the use of Diceware keys we enable the user to obtain a secure and easy to remember passphrase. Besides some existing challenges like initial password creation for all accounts is the passwordless solution practicable. Future work should focus on a more detailed analysis of our approach regarding security and the design of attacks on the proposed application.

BIBLIOGRAPHY

1. Hakbilen, O., Perinparajan, P., Eikeland, M., and Ulltveit-Moe, N. (2018). SAFEPASS-Presenting a Convenient, Portable and Secure Pass- word Manager. In ICISSP (pp. 292-303).
2. Li, Z., He, W., Akhawe, D., and Song, D. (2014). The emperor's new password

manager: Security analysis of web-based password managers. In 23rd USENIX Security Symposium (USENIX Security 14) (pp. 465- 479).
3. Bednarek A. (2019). https://www.ise.io/casestudies/password-manager- hacking/
4. Zhao, R., and Yue, C. (2014). Toward a secure and usable cloud-based password manager for web browsers. Computers and Security, 46, 32- 47.
5. Yee, K. P., and Sitaker, K. (2006, July). Passpet: convenient password management and phishing protection. In Proceedings of the second symposium on Usable privacy and security (pp. 32-43). ACM.
6. Halderman, J. A., Waters, B., and Felten, E. W. (2005, May). A convenient method for securely managing passwords. In Proceedings of the 14th international conference on World Wide Web (pp. 471-479). ACM.
7. Gaw, S., and Felten, E. W. (2006, July). Password management strategies for online accounts. In Proceedings of the second symposium on Usable privacy and security (pp. 44-55). ACM.
8. Gott A. (2014, September). The Scary Truth About Your Passwords: An Analysis of the Gmail Leak. https://blog.lastpass.com/2014/09/the- scary-truth-about-your-passwords-an-analysis-of-the-gmail-leak.html
9. Couillard K. (2015, September). Password Security Survey Results. https://roboform-blog.siber.com/2015/03/06/password-security-survey- results-part-1
10. Gasti, P., and Rasmussen, K. B. (2012, September). On the security of password manager database formats. In European Symposium on Re- search in Computer Security (pp. 770-787). Springer, Berlin, Heidelberg.
11. Billemont M. (2019). https://masterpassword.app/