Global Information Assurance Certification Paper

Security Considerations for Team Based Password Managers

GIAC GSEC Gold Certification

Author: Matthew Schumacher, schu@schu.net
Advisor: Chris Walker
Accepted: July 10, 2018

Abstract

Password management applications are a common and practical way to store complex passwords. They use encryption to protect the passwords from attack, but like in any other cryptographic system, they rely on a secret key to encrypt the data. The typical approach is to derive the secret key used to the encrypt the password database from a master password. This eliminates the requirement to store it or protect the secret key; however, this approach doesn't work well for multi-user password managers, as team based password management applications need to allow for each user having his/her own unique password, and may require other features such as password sharing, fine grained access control, or domain integration. This paper explores a few ways that different password management applications work in a team environment, and the strengths and weaknesses of their implementations. By learning about some of the underlying technologies and principles, then analyzing a few popular software applications, the reader should be better equipped to choose a solution that best fits their functionally and security requirements.

1. Introduction

Password managers are a popular and effective way to manage passwords. They give the user the ability to organize and store virtually unlimited numbers of complex passwords in an encrypted database that itself only requires one password or key to open.

As with any other cryptographic system, the strength of the system is directly related to the encryption algorithm used, and protecting the secret key that the algorithm uses to encrypt the data (Ellison, Schneier, 2000). Password managers typically use the master password or a derivative of the master password as the secret key, which eliminates the requirement to store and protect the key as it's entered by the user whenever the password database is opened and decrypted.

Requiring the user to effectively produce the secret key is a good way of protecting the key, as it's not written to disk, but typically doesn't work well for team-based password management. The problem is that each user has their own password that should be kept private from other users. This means that the password manager can't directly encrypt the password database with the user's password or a derivative of it, instead the secret key(s) must be stored on a server, or protected with another layer of encryption.

2. Why multiple users?

Password mangers can solve a lot of problems for teams of people that are responsible for the management and security of information systems. While it is always the best practice to assign a separate account and password for each user of each system

(Sans Institute, 2014), that isn't always possible or practical. Consider a few situations where there might be a requirement to store private information that a few people may need to know:

- Account information for a simple system that doesn't allow for more than one user such as a phone, a simple network device, a thermostat, or a conference bridge access code.
- Account information for a vendor website that doesn't allow for multiple users to access the same account.
- A local administrative account that bypasses a network user database so that access can be granted in the event that the network user database is unavailable.
- A private key for an SSL certificate that is used on multiple platforms or systems.
- The password for a secret key that is used to sign other keys.

2.1. The Secret Key Storage Conundrum.

Storing secret keys is not an easy problem to solve. The issue is that secret keys are required to decrypt the data they protect, so they must not fall into the hands of an attacker, while at the same time must be available to the system that accesses or encrypts the protected data.

Consider a web server on the Internet serving SSL web pages. That server has a public and a private (secret) key. When the web server is started it needs to know the secret key so that it can decrypt information sent to it, but should the web server become

compromised, the attacker may be able to gain access to both the data and secret keys which make all of the encrypted data visible to the attacker.

One possible solution is to store the secret key on the server and make it non-obvious how to gain access to the key (Matthew Mombrea, 2014) however, this doesn't work well because the web server configuration will point the attacker directly to its location.

Another solution is to not store the secret key on the server at all (Matthew Mombrea, 2014), but the issue with that is the web server software can't process encrypted data until it has the secret key. This means that if the web server is restarted, then the secret key (or a password which decrypts the secret key) will need to be provided manually by an administrator user or by some form of key management system. If provided by an administrator, restarting the web server software will require human intervention by someone with a memorized password, and if provided by a key management system, then the problem shifts to protecting the credentials used to get access to the secret key from the key management system.

2.2. Password Manager Secret Key Protection.

Because of the difficulties with protecting secret keys, most single-user password management programs store (or derive) the secret key from something the user recalls and provides, then when access to the password data isn't required anymore, they lock by closing the database and removing it from memory. **Keepass** erases all security-critical information from memory by overwriting the memory before releasing it back to the operating system (Dominik Reichl, 2018).

For additional security some single-user password management applications allow you to store a much more complex key on a piece of hardware, such as a thumb drive or use a hardware token such as a **yubikey** (Dominik Reichl, 2018). This effectively causes the password data to be protected by both a hardware token and a user memorized password.

2.3. Problems that adding multiple users creates.

Allowing multiple users to the access the same encrypted password data poses problems to the traditional single user password manager which typically uses the master password or user specific encryption to access the secret key.

There are typically three approaches to this problem. Using a shared master password and single password database, using a server-based solution that manages all of the keys and gives the users access to protected information through an encrypted tunnel, and using a combination of symmetric and asymmetric encryption to encrypt the password data and share the secret keys using a user's private key. For more information about symmetric and asymmetric encryption please see the following article:

- https://www.cs.utexas.edu/users/byoung/cs361/lecture44.pdf

2.3.1. Shared Master Password.

One method for allowing multiple users access to the same password data is to share a master password which is used to decrypt a shared password database. In this method a single user password manager is used with multiple users having the master password and access to the encrypted password database. This solution doesn't have any access control so it doesn't work in an environment where there isn't complete and total

trust of all users. For these reasons, it does not conform to password protection best practices (Sans Institute, 2014), and should be avoided.

2.3.2. Server Based Solutions

Another method for allowing multiple users to access the same password data is to store the password data in an encrypted database or datafile, then have a server component with access to the secret key decrypt the database give users the password data after they are authenticated.

This method allows the server to integrate into 3rd party authentication systems such as Active Directory and can even leverage the account recovery systems already in place such as resetting an account password. Very fine-grained access control can be implemented, but the access control is enforced by the server component, and not the underlying encryption. This means that there is no connection between who can access protected information and who has access to the secret keys used to decrypt that protected information. If an attacker is able to steal the encrypted password database and the secret key used to encrypt it then all of the password data will be compromised regardless if the attacker has a valid account or not. Protecting against this attack is difficult, because the secret key is usually stored on the server alongside the encrypted password database.

In this type of system, anyone that has physical access to the server, or access to it's backup, or is able to get administrative access to the server, will very likely be able to recover all of the password data it manages. This type of system is also vulnerable to attacks on the server component and any technologies it depends on.

2.3.3. Systems that use Symmetric Encryption for Password Data and Asymmetric Encryption for Sharing.

Systems that use asymmetric encryption for sharing typically encrypt the password data with symmetric encryption, then encrypt the resulting secret key with the public key of each user that has access to the password data. The private key for each user is then protected with symmetric encryption that is derived from the user's master password.

This hybrid approach requires the user's master password to decrypt the password data which means that the access control is implemented in the encryption itself. Users that do not have access to specific protected data aren't just denied access, they don't have the keys needed to decrypt the protected data. While access control is still used to protect access to the encrypted data, it's not critical to security, as the encrypted data is useless without the secret key.

Because user access to protected password data is tied to their private key and thus their master password, it's more difficult to integrate with other authentication systems such as Active Directory. Also, should a user forget their master password, there is no way to recover it, and any password that wasn't already shared with other users will be lost.

3. An in-depth look at several popular password managers.

3.1. Pleasant Password Server

3.1.1. Storage mechanism

Pleasant password server is a server-based solution that stores the password data in a single encrypted SQLite database. It can also use a PostgreSQL or Microsoft SQL server to store password data. In the case of SQLite, the secret key used to decrypt the database is stored as part of the connection string in an encrypted windows registry key (Caleb Mathison 2017). In the case of other back end databases, Pleasant uses the encryption and key management system built into whatever database you use (Caleb Mathison, 2018).

3.1.2. Features and benefits

The advantage of using a single key to protect the entire password database is that it's simple to implement and divorces the encryption and authentication. By using the back-end database encryption features and/or storing the secret key in the Windows Registry, the Pleasant Password Server doesn't do anything to protect the database secret key, it simply passes those responsibilities to the Windows Registry. By doing this, the running server always has access to all of the information in decrypted form, so managing which user has access to what protected information is nothing more than access control built into the Pleasant Password Server software. The advantage is that account recovery doesn't require decrypting data. Another advantage is that as long as the database connection string and database is backed up, the protected data can be recovered.

3.1.3. Authentication

Authentication on the Pleasant Password Server is completely controlled by the server application which has complete access to the encrypted database. This allows Pleasant to integrate with other systems that provide authentication services. Currently Pleasant integrates with OpenLDAP and Active Directory. Because an account isn't in any way tied to the encryption, resetting an account password will recover access to the encrypted password data.

3.1.4. Security considerations

All of the database encryption is performed on the server with the secret key stored in the windows registry. The integrity of the entire system and all of the data it manages hinges on keeping that secret key private, which is difficult because the key and data both reside on the server. If an attacker was able to gain administrative access to the server running Pleasant Password Server, they could use one of the many Windows registry decryption tools available on the Internet to get the database secret key, then use that data to open the SQLite database. Another attack would be to gain physical access to the server, or it's backup.

Another consideration is how access control is implemented. Access control is a function of the server software authorizing access to users and is not directly connected to the underlying encryption. A successful attack on the server software could reveal all of the protected password information is stores.

Pleasant Password Server uses a web client or a modified local client such as KeePass or Password safe. The modified local client connects to the server with the credentials of the user then the server creates a KeePass style password database using those credentials and sends it to the client. Because the user credentials are sent to the Pleasant Password Server using an encrypted web page, it is vulnerable to man-in-the-middle attacks.

3.2.1Password.com

3.2.1. Storage mechanism

1Password uses a combination of symmetrical and asymmetrical encryption to protect and share password data (AgileBits, 2017, pages 9-11). The password data itself is stored in an encrypted database 1Password calls a "Vault" using AES256 symmetrical encryption (AgileBits, 2017, page 17). Users also get a public/private key pair using RSA-OAEP asymmetrical encryption, which is used to encrypt the secret key for the vault (AgileBits, 2017, page 18). The private key for each user is protected with a password and account ID unique to each user. (AgileBits, 2017, page 19) When a user shares a vault with another user, the vault secret key is encrypted with the new user's public key and sent to them so that they can use their own private key to decrypt the vault secret key which then decrypts the vault (AgileBits. 2017, page 21).

3.2.2. Features and benefits

1Password's authentication and access control is enforced by cryptography instead of relying on software or personnel policy (AgileBits, 2017, page 6). The means that the software lacks what it needs to decrypt the password data unless the master

password is provided by the user, which can only decrypt data that the user has access to. This significantly reduces the attack vector as attacking the server system would only yield a lot of encrypted information that is useless without the secret keys of individual users.

1Password uses a combination of a password called the "Master Password" and an account ID called the "Secret Key" to protect each user's private key (AgileBits, 2017, page 9). This combination protects against weak passwords because guessing the password still wouldn't provide access to the user's private key unless the account ID is also known. Trying to guess or brute force the account ID is very unlikely as there are over 2^128 possible combinations, and because generating this account ID is completely random, unlike passwords typically used by humans (AgileBits, 2017, page 12).

3.2.3. Authentication

While all of the data in 1Password is useless without the password/account ID combination, authentication is still used to restrict which encrypted data is sent to the client. Because 1Password's guiding principles (AgileBits, 2017, page 6) requires that the encryption is what ultimately protects the password data, 1Password uses the encryption itself to authenticate users. The 1Password service sends a unique session key, encrypted with the user's public key to the client. The user provides the master password and account ID to the client, which then decrypts the session key. This proves they have the master password and account ID. Once decrypted, the session key is returned to the service encrypted with the public key of the service. This effectively authenticates both sides to each other as nobody else has access to the private keys, and the secrets needed to use them.

3.2.4. Security considerations

1Password has a cloud based server component, however, nothing stored in the cloud is sufficient to decrypt user password data. All of the decryption is done in the client, which makes it the preferred target of an attack. The client is implemented as a standard desktop application, and as a web client. The web client is sandboxed using chrome's mojo (https://discussions.agilebits.com/discussion/84078/1password-x-security). While this offers a good first line of defense, a web browser may be running other content and code that may try to attack the 1Password web client. This means that it might be possible to steal the secret key from the web client if a vulnerability is found in the browser. The native desktop client doesn't manage untrusted content and it also has access to some operating system security features like trying to forget the secret key when it's no longer needed, but is still vulnerable to attack using standard attacking tools like gaining access to system memory or a keylogger. (AgileBits, 2017, page 52)

3.3. Keepersecurity.com

3.3.1. Storage mechanism

Keeper also uses a combination of symmetrical and asymmetrical encryption to protect and share password data. Keeper encrypts each individual password with an AES256 key they call the "Record key". The record key is protected by another key they call the "Data Key" which is encrypted by the user's master password. Another key called the Client Key is used to encrypt the data at rest on the user's device and it is also encrypted by the users master password.

Each user also gets an RSA public/private key pair which is used for sharing passwords. When a password is shared with a new user, the record key is encrypted with the new user's public key and sent to them. Keeper also supports having a shared folder of keys. In this case the shared folder itself has a secret key, which is used to encrypt the record key which protects the password data. When a folder is shared, the secret key of the folder is encrypted with the public key of the user that is being granted access.

3.3.2. Features and benefits

Keeper calls itself a "Zero-Knowledge" security provider, which means that Keeper doesn't have the secret keys needed to decrypt password data. Access control is enforced in the cryptography itself.

Keeper has an emergency access feature which allows a user to share their password data with another Keeper user, but not until a specified wait time has elapsed. Keeper's documentation doesn't say how this is enforced.

The Keeper native clients use certificate pinning to block against man-in-the-middle attacks. This means that the Keeper client will only connect to servers that have a certificate it already knows about.

3.3.3. Authentication

Authentication to the normal Keeper service is performed by the client providing the server with a hash derived from the master password and the server comparing that hash to a known value. This allows authentication without transmitting the master password to the Keeper servers. The hash is generated using the Password Based Key Derivation Function 2 which is part of the RSA Public-key Cryptography Standard.

Keeper also has a single sign on (SSO) feature for business customers. It uses a software application called Keeper SSO Connect that is installed in the business infrastructure. This software manages all of the encryption keys for the end users and provides the secret keys needed to decrypt protected password information when the user authenticates.

This has the advantage of allowing an administrator to reset passwords so that users can regain access after forgetting their master password, or synchronizing the master password to their domain password, but the disadvantage of having a server in the network that has all of the master password keys stored on a disk alongside with other information needed to use them.

3.3.4. Security considerations

Keeper has a cloud component that stores the encrypted password data, however the data needed to decrypt the data is never transmitted to the cloud, as the decryption happens on the client. The client is implemented as a native client and as a web client. The native client stores all of its local data in encrypted format, so it is most vulnerable to typical application attacks such as keyloggers, or looking for secret keys in system memory. The Keeper documentation doesn't mention using a browser sandbox for the web client, so it's not disclosed how the web client protects data it manages.

If using Keeper in SSO mode, then all of the keys are managed by the Keeper SSO connect software. This fundamentally divorces the secret key from the user's master password, and thus makes the Keeper SSO connect software the focus of any attacks. If the software was compromised, the attacker would have all of the secret keys needed to decrypt password data. The Keeper documentation doesn't describe how

Keeper SSO connect protects the user's secret keys, only that it stores them on a locally installed server. This could make Keeper protected passwords vulnerable in the case where an attacker gains access to the server or it's backups. (https://keepersecurity.com/user-guides/guide-admin-console.html#configure_ldapad).

The Keeper SSO Connect software uses Security Assertion Markup Language (SAML) 2.0 to authenticate users. According to the SAML 2.0 standard, the authentication is performed by the Identity Provider directly, which means that the Keeper SSO software never sees user passwords. https://developers.onelogin.com/saml

4. Conclusions

There are many different password managers available in the market today and all of them claim to keep your password information secure, however they each have their own distinct features and security compromises which can usually be discovered when reviewing the products security white paper or inferring from any implementation documentation available.

Some password managers are very forthcoming about how they are implemented such as 1Password, while others have very little information. This can make it easy or difficult to understand how the software manages secret keys, which is extremely important when considering the security of the software or solution, and which attacks it might be vulnerable to.

Some password managers can operate in several different modes such as Keeper Security, which protects secret keys differently depending on if the SSO software is used

to integrate with another identity provider or not. In this case the documentation might lead one to believe that the secret key that protects the password data is always derived from the Master Password, but in case of SSO integration, that is not true.

Single Sign On integration fundamentally disconnects the user's password from the secret key used to protect the data, so it's important to find out exactly how the secret keys are protected, then through a risk assessment determine if the risks are acceptable.

Matthew Schumacher, schu@schu.net

References

Ellison, Carl; Schneier, Bruce. (Volume 16, Number 1, 2000) Ten Risks of PKI: What You're not Being Told about Public Key Infrastructure. Retrieved from https://www.schneier.com/academic/paperfiles/paper-pki.pdf

Sans Institute. (2014) Password Protection Policy. Retrieved from https://www.sans.org/security-resources/policies/general/pdf/password-protection-policy

Mombrea, Matthew. ITworld (2014). A basic encryption strategy for storing sensitive data. Retrieved from https://www.itworld.com/article/2693828/data-protection/a-basic-encryption-strategy-for-storing-sensitive-data.html

Reichl, Dominik. (2018). KeePass Help Center, Security Retrieved from https://keepass.info/help/base/security.html

Reichl, Dominik. (2018). KeePass Help Center, Composite Master Key Retrieved from https://keepass.info/help/base/keys.html

Reichl, Dominik. (2018) KeePass Help Center, KeePass & YubiKey Retrieved from https://keepass.info/help/kb/yubikey.html

Mathison, Caleb. (2017). Pleasant Solutions Public Documentation, Service Configuration Utility Retrieved from https://info.pleasantsolutions.com/Documentation/Pleasant_Password_Server/B._How_to_configure_Pleasant_Password_Server/5)_Service_Configuration_Utility

Caleb Mathison. (2018). Pleasant Solutions Public Documentation, Find a Connection String Retrieved from https://info.pleasantsolutions.com/Documentation/Pleasant_Password_Server/B._How_to_configure_Pleasant_Password_Server/4)_Changing_Databases_for_Pleasant_Password_Server/Find_a_Connection_String

AgileBits. (2017-04-12, version 0.2.6). 1Password Security Design Retrieved from https://1password.com/files/1Password%20for%20Teams%20White%20Paper.pdf.

Matthew Schumacher, schu@schu.net