Security Analysis of Dashlane

Paolo Gentili, Sarah Shader, Richard Yip, Brandon Zeng

May 2016

Abstract

We perform a security analysis of Dashlane, a password manager and digital wallet with over 3 million users. Given the sensitive data associated with such an app, including passwords, credit card numbers, and other personal information, any vulnerabilities in the app could have significant consequences for its users. Throughout our attacks, we find only minor vulnerabilities in Dashlane, which we present in this paper along with recommendations and other future potential avenues of attack.

1 Introduction

With the vast number of online accounts that most people have in the digital age, many people have turned to password managers to help remember all of their passwords. Our group has performed a security analysis of one such app: the password manager Dashlane. Dashlane is a password manager and digital wallet app available on Mac, PC, iOS and Android. First released in 2012, it has amassed over 3 million users and has quickly grown as a way for users to not have to remember passwords for accounts on many different websites, such as Facebook, Google, and PayPal.

Dashlane keeps a record of all the different passwords for a user protected by the user's "master password," meaning that the user only needs to remember their master password in order to access their other passwords. Dashlane also offers a premium feature which syncs passwords for a user across different devices (i.e. from iOS to Mac) as well as the option to share passwords with other Dashlane users, either with full access (the recipient sees the password) or limited access (the recipient can use the password to log in, but does not actually see the password). In addition, Dashlane has features which assess the security of passwords, offering to automatically update the passwords if they are insecure or out of date, as well as features which randomly generate "strong" passwords.

Since Dashlane, as a password manager, is responsible for protecting a lot of sensitive information, it is vital that Dashlane is very secure. There has not been a comprehensive security analysis of Dashlane to date. However, the 2014 paper *The Emperors New Password Manager: Security Analysis of Web-based Password Managers* offers a security analysis of five different password managers: LastPass, RoboForm, My1login, PasswordBox, and NeedMyPassword [6]. This paper reveals vulnerabilities in all of these password managers with a variety of different root causes ranging from web vulnerabilities such as CSRF and XSS to authorization mistakes in features such as password sharing. For these reasons, we chose to investigate the security of Dashlane by applying some of these same attacks.

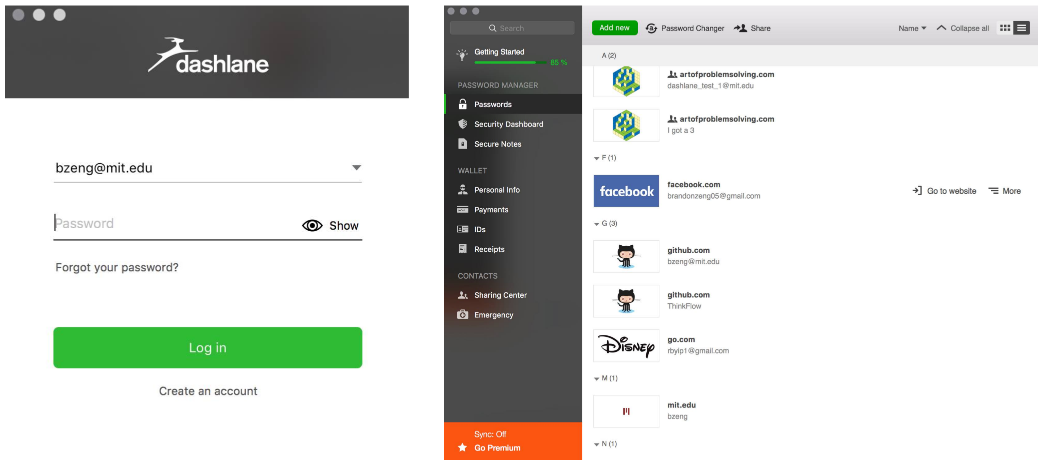


Figure 1: The Dashlane log-in window and desktop app

2 Structure of Dashlane

When a user creates a Dashlane account, they use their email and choose a "strong" password defined as having at least eight characters, one uppercase letter, one lowercase letter, and one number. The user can now log in to Dashlane from its web or desktop app from the user's computer, or from the user's phone. Whenever the user attempts to log in to their account from a new device, the user must enter a six-digit code sent by email in addition to their master password, offering another layer of authentication.

With the help of a browser extension, Dashlane automatically detects existing passwords that have been saved to the browser and adds them to the user's list of passwords. When a user logs into an account that has not yet been saved, Dashlane (with the help of the extension) will offer to automatically save that password to the user's account. The Dashlane extension also allows a user to automatically log in to any of their accounts that have been saved. A user may also manually add a username and password to their Dashlane account, and this account can then be used in the same manner as any other saved account.

Dashlane uses AES-256 with CBC to encrypt user data, using a key derived from the master password from using more than 10,000 rounds of PBKDF2 with a random 32-byte salt. From examining the source code of the Dashlane web app as well as the Dashlane Whitepaper [1], we verified that the encrypted Dashlane data is formatted in the following way:

- The data is represented in Base64
- Once decoded to binary or hexadecimal, bytes 1 through 32 are the salt
- Bytes 33 through 36 represent whether the data is compressed or not
  - If these bytes represent 'KWC3' in ASCII, the data is compressed
  - Otherwise, the data is uncompressed
  - Based on the compression, there is a different protocol for selecting the AES key and the initialization vector

Dashlane uses OpenSSL's PKCS5_PBKDF2_HMAC1 function with 10204 iterations with the indicated salt to produce a derived key from the user's master password. Dashlane then uses OpenSSL's EVP_BytesToKey function taking the previously generated key and salt, using SHA and only 1 iteration.

These values can now be used to decrypt the Dashlane data. If the data is compressed, we use the first key we generated with the EVP_BytesToKey initialization vector. Otherwise, we use both the key and initialization vector generated by EVP_BytesToKey. Once decrypted, the data is in XML form and the various data types are summarized in [7].

3 Responsible Disclosure

The research in this paper was performed with the permission of the company under a responsible disclosure policy. Our analysis audits Dashlane's web application, https://www.dashlane.com/app/, and its desktop app, version 4.1.1. Some of the security issues discussed below may affect Dashlane's service, so we are currently contacting Dashlane and will delay the public posting of this paper by four weeks while waiting for a response. We will further delay its posting upon request by Dashlane.

4 Attack Model

We mainly consider attackers who are normal users of Dashlane as opposed to insiders, such as Dashlane employees, since it is very hard to reason about the resources and permissions that insiders would have, other than that they should be unable to access user information. We focus on attacks that reveal a user's private information instead of attacks such as denial of service attacks. We assume that attackers only have access to what is publicly available from Dashlane, such as their web app and Chrome extension source code as well as any other public encryption software such as OpenSSL. We also assume that the attacker potentially has access to a lot of computing power, which can be easily obtained from resources such as Amazon Web Services, as well as a lot of time and patience.

5 Vulnerabilities

5.1 Web

We began by considering the public code Dashlane has available, using Google Chrome's Developer Tools to identify scripts and monitor local storage, as well as examining the Chrome extension's code. We were unable to find hardcoded values or unencrypted information that would make Dashlane vulnerable.

We then proceed to use OWASP's Zed Attack Proxy, a penetration testing tool, which revealed minor vulnerabilities but nothing that would compromise user accounts. This was further confirmed when we attempted to use a XSS attack by entering Javascript into forms on the web app, but Dashlane sanitized our inputs and the redirected URL had all Javascript escaped. Similarly, attempting a CSRF attack was unsuccessful as Dashlane appears to use secure random tokens in its HTTP requests.

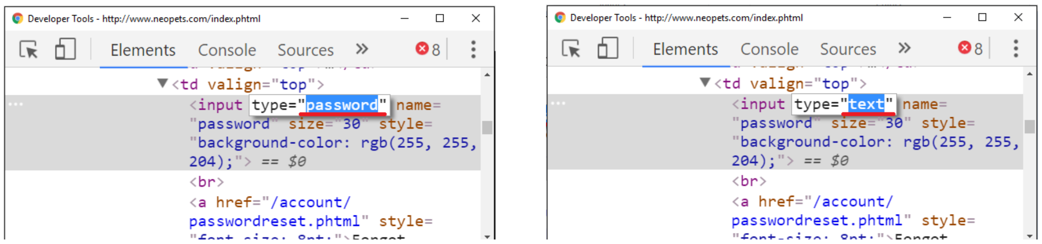
Finally, we used the MITMProxy tool for analyzing network traffic between the mobile app and Dashlane servers. To use this man-in-the-middle approach, we installed a malicious root certificate on our phone, which allowed us to encrypt and decrypt traffic to and from the device. Using MITMProxy, we were able to log API calls and view the transmitted content, which notably included an encrypted record of the account's passwords. Based on [7] and given the user's password, we would theoretically be able to decrypt the record and view that user's stored passwords.

5.2 Features Testing

We decided to test Dashlane's password sharing feature, in particular the claims regarding passwords shared with limited rights. If one chooses to share a password with full rights, the recipient can view the password, edit the password, share it with others, and even revoke access from the original owner. Instead, if a password is shared with limited rights, the recipient should still be able to log in to the account through the help of a browser extension, but he or she should not be able to view the password, edit it, or share it with others.

Even with a password shared with limited rights, we realized that the Dashlane browser extension would eventually have to handle unencrypted password data in order to log in to a website. Thus, we felt that there might be an opportunity for an attack for the recipient with limited rights to a password to obtain the password's plaintext.

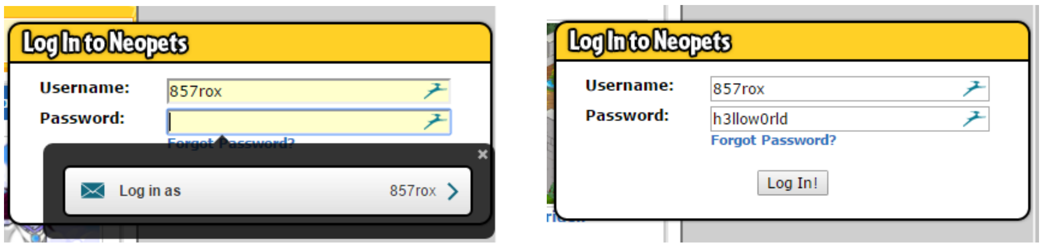
This intuition turned out to be true. We first went to the log-in page corresponding to the shared password and used Google Chrome's "inspect element" on the password field. Here, we changed the field from a "password" type to a "text" type, so any text entered into that field would show up in plain text rather than as password bullets (see Figure 2). At this point, we used the Dashlane web extension to automatically log in to the account and were able to view the plaintext of the password that had originally been shared with limited rights (Figure 3).



```
▼<td valign="top">
   <input type="password" name=
   "password" size="30" style=
   "background-color: rgb(255, 255,
   204);"> == $0
   <br>
   <a href="/account/
   passwordreset.phtml" style=
```

```
▼<td valign="top">
   <input type="text" name=
   "password" size="30" style=
   "background-color: rgb(255, 255,
   204);"> == $0
   <br>
   <a href="/account/
   passwordreset.phtml" style=
```

Figure 2: We change the password field to a text field using Chrome Developer Tools



Username: 857rox
Password:

Username: 857rox
Password: h3llow0rld

Figure 3: Logging in with the Dashlane extension reveals the password in plaintext

This vulnerability compromises several of Dashlane's promises regarding passwords shared with limited rights. Obviously, this violates the claim that passwords shared with limited rights cannot be viewed by the recipient. Once the recipient has access to the password in plaintext, he or she can now also share access to the corresponding account with other unintended recipients. Additionally, since many users use the same passwords across different accounts (despite the efforts of password managers to combat this), this vulnerability may compromise other accounts of the user that were not shared but have the same password.

5.3 Brute Force

The Dashlane website and desktop app both require an email and password for login. During testing, we discovered that neither login forms limit password retries. Furthermore, we were able to determine emails linked with dashlane accounts (the email address is the username for Dashlane accounts) as the UI displays “Incorrect login” for nonexistent usernames but “Incorrect password” for existing ones. This means an adversary could potentially brute force usernames and passwords.

However, this threat is mitigated since Dashlane uses Two-Factor authentication to authenticate new devices. If an attacker tried a brute force attack on a specific account, the attacker would first have to authenticate the account on the specific device. As mentioned earlier, whenever a Dashlane account is accessed on a new device, the user is sent a six-digit security code which expires after three hours. Therefore, an attacker would have to brute force both the security code and the password. Additionally, the security code resets after five incorrect attempts. While this makes a brute force attack on the password much harder, it is not completely infeasible, and we should note that six digits is abnormally small. Furthermore, if the attacker knows the password, brute-forcing a six-digit security code–even one that changes every five attempts–will take about $70,000$ attempts to get a 50% probability of succeeding. While large, that number of attempts is not infeasible.

5.4 Device Authentication

As mentioned in the introduction, Dashlane offers device authentication as an additional layer of security. To prevent a foreign adversary from brute-forcing a user’s password, Dashlane requires that all computers and browsers first be registered with the user’s account before logging in. This entails sending an email containing a six-digit security code that the user must enter in addition to the master password. Once the device or browser is registered, the user is free to log in without the security code.

Our group focused on trying to bypass this device registration requirement. We found out how Dashlane authenticates devices and what is necessary in order to forge device registration.

We first considered the possibility that Dashlane used MAC addresses to authenticate computers. We created and authenticated an account on one computer and then spoofed that machine's MAC address on another. However, when we attempted to log in to the same account on the second computer, we were still prompted to enter a security code, meaning that Dashlane did not (solely) use MAC addresses for device authentication.

Since this approach did not work, we attempted to determine where authorization occurred on an already registered device in order to see if we could forge this authorization and bypass the device registration phase. We were eventually able to determine where the device authorization took place. Since login for registered devices can occur offline, the authorization could not happen on Dashlane servers. After some hard drive searching, we found a folder containing folders that corresponded with every account registered with the computer: the name of each folder matched the emails of all the computer's registered accounts.

In the folders contained AES-encrypted files that contained passwords and login info, sharing rights details, purchase histories, and more. These files are synced with Dashlane servers upon login (provided there is internet access). In particular, we found that the file *localSettings.aes* contains the device authentication. Note that this device authentication is NOT tied to the computer itself; for example, if this folder was copied onto another computer, that computer could log into the account without prior verification.

To confirm this, we created two Dashlane accounts with usernames *dashlane1@mit.edu* (which we will call account A) and *dashlane2@mit.edu* (account B) and gave them the same password (see Figure 4). Next, we logged into account A on one computer (called computer a) and account B on another (computer b). Finally, we created a folder called *dashlane1@mit.edu* in computer b—which did not have registered access to that account—and inserted the encrypted *.aes* data from account B's folder. Finally, we took

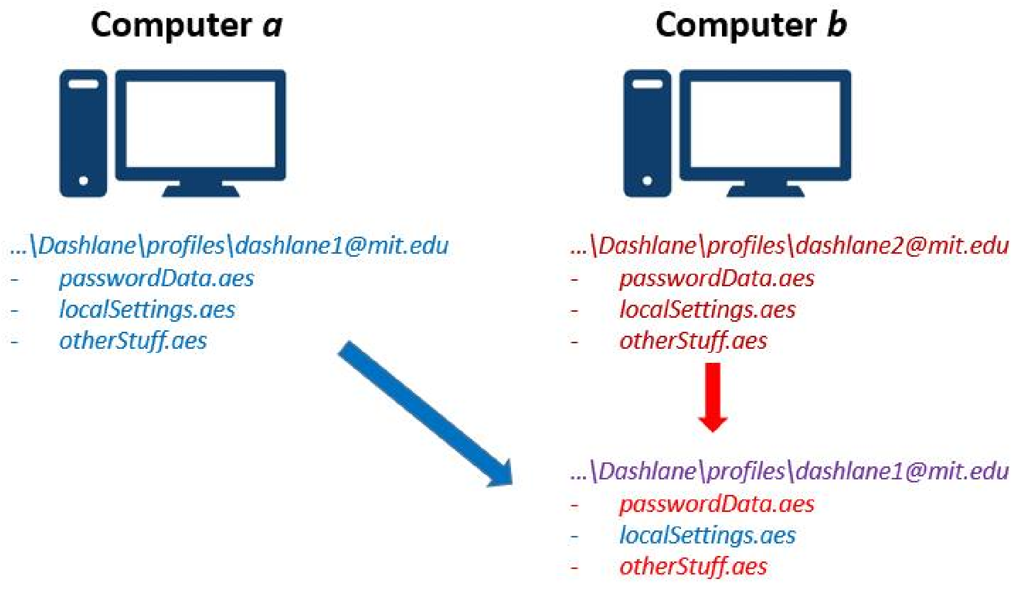


Figure 4: By combining the contents of two account's folders on two different computers, we were able to forge device authentication.

the *localSettings.aes* from account B's folder and replaced the existing file in this new folder.

We found that we were able to log into account A on computer b. However, it displayed all the password data from account B since all the *.aes* password data was copied over from account B and unencrypted with account B's password (which is the same as account A's password). We were able to sync account A with the Dashlane servers, though, which resulted in access to all of account A's password data.

Thus, with access to account A's password and *localSettings.aes* file, we were able to bypass the device registration requirement (see Figure 4). It turns out that, with limited reproducibility, we were able to get the *localSettings.aes* file by creating an empty *dashlane1@mit.edu* folder, logging into the account, and letting Dashlane sync. Because of this, Dashlane's device authentication feature is suspect at best.

6 Recommendations

Most of our recommendations revolve around the potential vulnerabilities found.

First of all, we recommend that Dashlane provide accounts with a username other than the user's email address. That way, it prevents adversaries from finding out whether certain email addresses have Dashlane accounts. We also recommend that Dashlane implement a maximum number of unsuccessful login tries to discourage brute-forcing passwords. In the same vein, the device authentication security code should be longer than six digits.

The limited rights password sharing idea is nice, but practically impossible to implement correctly in its current state. The reason for this is that Dashlane is at the mercy of the web browsers: if a web browser (e.g. Chrome) allows users to inspect and edit the HTML and CSS on webpages, it becomes very difficult to give a user access to an account without revealing the password itself. Thus, we recommend that Dashlane remove the limited rights password sharing feature.

Finally, the device authentication feature should not solely rely on a file appearing in a folder. Instead, it should require that a computer's MAC address also match up. While spoofing a MAC address is not impossible, companies like Intel are making chips that do not allow users to change their MAC addresses. This would make it more difficult to forge the device authentication file.

7 Future Work

Due to limited time and Dashlane's numerous features, we did not have the opportunity to fully test Dashlane for security vulnerabilities. For future work, we would like to pursue some of the leads found with brute force attacks and fake device authentication. In addition, we would like to investigate the other features of Dashlane. In particular, we believe that the in case of emergency sharing and the password generating features hold potential vulnerabilities. Additionally, for future work we want to investigate potential ways to improve Dashlane such as adding the option for master password recovery.

7.1 In Case of Emergency Sharing

Dashlane offers a feature to share your data with an emergency contact in the case of an emergency. If user A grants user B access in case of emergency, user B can request access to user A's data, and if user A does not respond within a certain amount of time (predetermined by user A), user B receives access to all of user A's data. Depending on how this is implemented, this may indicate that Dashlane servers have access to unencrypted data, or more likely user A's data encrypted using a derivation of user B's master password. This would mean that user A may temporarily have access to a derivation of user B's master password, which could potentially decrypt some of user B's data. Conversely, the Dashlane server may have user A's data, which could potentially be decrypted by user B without the permission of user A.

7.2 Password Generator

Like many password managers, Dashlane offers a password generator which claims to generate "strong" and "random" passwords. The user can specify the length of the password as well as various other features such as whether the password should contain numbers, symbols, or be pronounceable. We were unable to find details about how Dashlane implements this "random" password generator, but in future work, details about the implementation of this password generator could potentially lead to more efficient brute force attacks on these generated passwords of Dashlane users.

Another notable feature is that Dashlane claims to store all previously generated passwords created by an account. This is intended for a user to potentially recover a previously generated password if they forgot to initially store it as a password in Dashlane. However, if these passwords are incorrectly stored, for instance if they are cached (unencrypted) somewhere, it would make it easy for an attacker to hack any recently created accounts made by Dashlane users. Clearly, if the password generator had some vulnerability, it could have major consequences for Dashlane users who rely on this feature.

7.3 Master Password Recovery

Out of the many features that Dashlane offers, it does not offer an option for master password recovery (which we unfortunately discovered in our early testing of the application). Most websites offer password recovery or password reset through email. However, since Dashlane in many cases actually stores the user's email password, this may not be a feasible option. In fact, such a password recovery scheme might even be detrimental in situations where the user's email password was compromised. Nonetheless, a password recovery option would make Dashlane more user friendly, so it would be worth investigating the possibility of creating a password recovery option.

8 Conclusion

Overall, we found Dashlane to be quite secure against all of the attacks that we carried out. All essential user data is encrypted using AES-256 with a key that derives from the user's master password, and neither the master password nor the key is stored locally or on Dashlane's servers. As a result, the security of AES ensures that it is infeasible to obtain a user's sensitive information without knowledge of their master password. Furthermore, even with the knowledge of a user's master password, it is difficult—though not impossible—to bypass Dashlane's device authentication to gain access to their account.

We ultimately found a vulnerability regarding Dashlane's limited rights sharing feature, which we recommend be removed to avoid giving users a sense of security that does not exist. We also discovered several practices that could be adjusted to ensure greater security, specifically against brute force attacks. While we have some ideas for future approaches to attack Dashlane, we did not discover any major vulnerabilities, which is a good sign for a security-based app like Dashlane.

9 Acknowledgements

We would like to think Professor Rivest and the rest of the 6.857 staff for their support and guidance. We would also like to thank Dashlane for giving us permission to research their product.

References

[1] "Dashlane Security Whitepaper" (2016). URL: `https://www.dashlane.com/download/Dashlane-Security-Whitepaper-V2.8.pdf`.

[2] *Dashlane Website*. URL: `https://www.dashlane.com/security` (visited on 04/05/2016).

[3] Nikolay Derkach. "A Tutorial for Reverse Engineering Your Software's Private API: Hacking Your Couch" (2015). URL: `https://www.toptal.com/back-end/reverse-engineering-the-private-api-hacking-your-couch` (visited on 05/02/2016).

[4] Geoffrey A. Fowler. "A Quick Fix for Poor Passwords" (2014). URL: `http://www.wsj.com/articles/a-quick-fix-for-poor-passwords-1418126603` (visited on 04/05/2016).

[5] Paolo Gasti and Kasper B. Rasmussen. "On The Security of Password Manager Database Formats" (2012).

[6] Zhiwei Li et al. "The Emperors New Password Manager: Security Analysis of Web-based Password Managers" (2014).

[7] *Notes about Dashlane*. 2014. URL: `https://gist.github.com/roblabla/4a43d0a5fc3769c815ab` (visited on 04/16/2016).