Client-side Attacks on the LastPass Browser Extension

Derk Barten
Master Security and Network Engineering
University of Amsterdam

February 2019

Abstract - This paper discusses client-side attacks to the LastPass extension for the Chrome browser. Two approaches are discussed: file system based and memory based attacks. The file system based attacks cover attacks that use files on the machine of the client to obtain their credentials. The memory based attacks consider attacks that use either live memory or memory dumps to obtain credentials. We have discovered that the locally stored vault can be decrypted when the remember password function is enabled. Furthermore, we are able to obtain the encryption key from a memory dump of the LastPass extension; this can be used to decrypt the local vault as well.

1 Introduction

In the past decade, the number of web-services has increased significantly. A part of these services contain important assets of the user, thus requiring a form of authentication to access. This is generally implemented as a username and password. Because good password hygiene dictates that an unique password should be used for every service, the user may end up with a large number of passwords that need to be remembered. As a result of the number of passwords, people often create simple, easily memorizable passwords and reuse passwords from different services [1]. Password reuse is dangerous as a compromised password for a single service may result in other services becoming compromised as well.

Password managers provide a solution to weak passwords and password reuse by using a single strong master password. A password manager stores the credentials of the services in a digital vault protected by the master password. The reduced cognitive load of multiple passwords allows one password to be long and complex. The passwords stored in the vault can be long and random considering the user does not have to remember these. The downside of a password manager a single point of failure; if the master password is compromised, the attacker obtains access to all accounts of the user. As an added layer of protection, Multifactor Authentication (MFA) can avoid stolen credentials after a compromised master password. With MFA enabled, the user requires a secondary device to authorize access to the vault after the master password is given.

Our research aims to discover in what ways an attacker can gain access to the vault of an user from the client-side. We will be using the LastPass password manager as case study. With 13 million users, the Lastpass password manager is one of the most prevalent password managers in the market. LastPass stores an encrypted form of the user's vault on their servers, to allows the user to login to their vault from any device. Our research will focus on specifically on the LastPass browser extension.

2 Research Question

In this research, we will evaluate the attack vectors of a possible attacker to obtain user credentials from the LastPass browser extension. Besides theory, we

are interested to show proof of concepts attack on a password manager. We state the research question as follows:

What client-side attacks can be used on the LastPass extension for the Chrome browser?

To put the research question more concretely, we will look at two types of attacks: *File system based attacks* and *Memory based attacks*. The file system based attacks will investigate how files on the file system may be used to compromise the password vault of the user. With this approach, a attacker may, for instance, look at locally stored files of the LastPass extension to find valuable data. The memory based attacks will investigate whether there is confidential information in the memory of the LastPass extension. This analysis of the memory is either performed live while the extension is active or from a memory dump that was taken while the extension was active.

2.1 Related Work

Li et al. [2] performed a security analysis of five popular browser-based password managers. For 4 out of 5 password managers, the attacker could extract user credentials for arbitrary websites. They found vulnerabilities a range of features, such as Bookmarklets, Onetime Passwords and shared passwords. These were caused by faulty logic, Cross-site scripting and Cross-site request forgery, among others. This research shows that the security of browser-based password managers is far from perfect. They have approached the browser-based password managers from the perspective of an outside attacker, we will evaluate the attacks from within the system of the user.

The LastPass extension has been the subject of research multiple times, resulting in the publication of various vulnerabilities. For instance, there have been found weaknesses in the auto fill functionality of lastpass [3] [4]. Furthermore, there were client-side attacks on the extension where websites could, under certain circumstances, inject JavaScript code in the extension [5].

Our research has accidental overlap with previous research conducted by Martin Vigo and his colleague Alberto Illera in 2014 [6]. They have performed client side attacks and have used the same method for the extraction of the master password from the Lastpass database as our research. It appears that the overlap in research expends further than we initially expected. Although their blog did not exactly specify the approach used to decrypt the locally stored accounts, we later discovered that their approach in their metasploit module appears to be very similar to our approach. Because both our research has multiple similarities, this presents an opportunity to discover whether LastPass has changed their security model in the years between our studies.

Regarding analysis of memory, this is a field that has been actively researched in Forensic science. The extraction of passwords from memory has been studied before [7], [8], [9]. There exists a plugin for Volatility from 2016 that is able to extract credentials from a memory dump of the LastPass extension [10]. This approach relies on a JSON file in the memory space of the extension that contains plaintext usernames and passwords. We were not able to reproduce these results; it appears that the architecture of LastPass has changed after this was published as passwords are no longer stored in plaintext.

3 Scope and Limitations

In this research is limited to the LastPass extension for the Google Chrome browser. LastPass was chosen because we observed that it is currently one of the most popular browser-based password managers; Chrome was chosen for the same reason, as it is the most popular internet browser. We will limit the research to the Microsoft Windows 10 operating system as Windows is currently the most used desktop operating system in the market. Although browser-based password managers are operating system independent, the methods for attacking will very likely vary between operating systems due to different security models.

The attack scenario we have chosen to investigate is relevant for Red team operations. In security assessments, a Red team may be tasked to gain access to a server of a certain company. If the server is diffi-

Software	Version
VirtualBox	6.0.0 r127442
Google Chrome	72.0.3626.81
LastPass Extension	4.24.0
Windows 10	17763.253
WinDbg	1809

Table 1: List of software and their respective versions that were used during this research

cult to infiltrate directly, a different approach may be chosen. Red teams may try to break into the machine of one of the administrators as a jumping point to the high security machine. On the machine of the administrator, it is not uncommon to encounter a password manager; this may contain credentials to the server. There are currently two methods that Red teams use when the KeePass password manager is encountered. First, the *KeeThief* tool [11] [12] can be used to use code injection in the process to decrypt the encrypted passwords in memory. The second and less complex approach, is the termination of the KeePass process together with the installation of a keylogger. We aim to record the attacks that can be used by Red teams for the LastPass extension.

3.1 Research environment

The research and experiments are performed on the Window 10 operating system inside VirtualBox on a GNU/Linux system. We will use the Google Chrome browser together with the LastPass browser extension. The specific versions are listed in Table 1.

4 General approach to reverse engineering the LastPass extension

Extensions for the Chrome web browser are written in the common web languages: HTML, CSS and JavaScript. In Windows 10, the code that forms the extension is stored, by default, under the `Extensions` folder in the Google Chrome UserData. When the browser is started, the locally stored code of the extension will be run. We observed that, when the page that contains the account credentials is requested, only a limited number of JavaScript files are called and executed. This excludes the backend of the extension, which invokes the connection with the database and the encryption/decryption of the credentials, among other tasks. In order to debug the backend parts, we would need to invoke their respective JavaScript files. Fortunately, with extension development, it is common to have a file called `background.html`. This file invokes all scripts that run in the background of the extension, this includes the backend. When this page is called, the Chrome developer tools can be used to show important JavaScript variables, such as keys and usernames, among others. By changing Chrome to developer mode, the extension could be edited to include debugging print statements without triggering Chrome to mark the extension as corrupt. Furthermore, breakpoints can be included in the backend parts to inspect the content of the JavaScript variables during runtime.

5 File system based client-side attack using the remember password function

This attack is based on the options to remember the password in the LastPass extension. When remember password is enabled, the password will be stored somewhere on the filesystem. If the password can be found, this can be used to gain access to the victims accounts.

The Lastpass extension, by default, keeps an SQLite database in the Chrome UserData under `Default\databases`. This database contains various tables: *LastPassData*, *LastPassPreferences*, and *LastPassSavedLogins2*, among others. The most notable of these tables is the saved logins table, which contains the fields: *username*, *password*, *last_login* and *protected*. This database is displayed in Table 2. In the username field, we see the username that we used to login to the extension in plain text. The password field contains two base64 encoded strings sepa-

rated by a pipe. The *lastlogin* field is a timestamp. Lastly, as we understand it, the protected field indicated whether the password is stored in the password field. For the account `victim_alice@outlook.com`, we disabled the option to remember the password. As a result, the password field is empty and the protected field is set to 0. Thus, we may conclude that the password will be stored in the saved logins table when remember password is enabled.

Vigo et al. [6] state that the first base64 encoded string of the password field is the IV and the second part is the encrypted password. We can verify that this statement is correct by inspecting the Javascript code. In the `enccbc` function under `lpfulllib.js`, we can confirm that the password field is split in two parts. A part of this function, as displayed in Figure 1, describes how data is encrypted and stored using the Advanced Encryption Standard (AES). Between the exclamation mark and the pipe, the output of `AES.eb64(c)` is inserted, which is the base64 representation the IV. After the pipe, we can see the specific parameters for AES, namely 256 bit key and CBC mode, which confirms the information that is listed on the website of LastPass. This allows us to conclude that the first 16 bytes is the IV, base64 encoded. The second part after the pipe is the encrypted data with AES, 256 bits key, Cipher Block Chaining mode.

```
var l = AES.CreateIV(16),
    c = l.str,
    u = l.arr;
i = "!" + AES.eb64(c) + "|" +
    AES.Encrypt({
        key: g_aKeys[r],
        iv: u,
        data: e,
        b64: !0,
        bits: 256,
        mode: "cbc"
    });
```

Figure 1: The specification of the AES encrypted field format. This snippet is located in the *enccbc* function in *lpfullib.js*.

5.1 Masterpassword decryption key

At this stage, the encrypted form of the master password is known. To proceed, we would require the master password to be decrypted. The next step is to find what key was used to encrypt the master password, as the IV and the ciphertext are already known.

Inside the *loginOffline* function in *server.js*, the snippet displayed in Figure 2 can be found. Using debug statements, it became clear that the variable r contains the username and o contains the encrypted data in the password field. The *lpdec* function decrypts the first argument with the key in the second argument. It appears that the password is encrypted with the SHA256 sum of the username as encryption key. We can confirm that this is correct by using the values that were encountered during the debug

```
if (2 == a[0].protected) {
        return void n(lpdec(o,
            AES.hex2bin(SHA256(r))));
}
```

Figure 2: Code snippet of the method used to decrypt the master password

username	password	last_login	protected
victim_bob@protonmail.org	!AMhA9punOT1FtPaZY4rkWA==|Y8T7eY4ojTdRSetkZClwaUkLy//bZIDfV7FADyr3UFE=	1548087135857	2
victim_alice@outlook.com		1548237278532	0

Table 2: This table shows in what format the saved logins are stored in the local database.

Username	victim_bob@protonmail.org
Key:	f2 31 c4 10 3e 56 9f 8e 4d 6f 3c 0c 8b 52 be 14 c4 58 1f f3 c8 5e 71 54 31 61 0a 14 64 b0 60 c5
Password Ciphertext	63 c4 fb 79 8e 28 8d 37 51 49 eb 64 64 29 70 69 49 0b cb ff db 64 80 df 57 b1 40 0f 2a f7 50 51
IV	00 c8 40 f6 9b a7 39 3d 45 b4 f6 99 63 8a e4 58

Table 3: Example of values encountered during debugging of the login function of the LastPass extension

session, given in Table 3.

Using the encryption key, IV, ciphertext with AES CBC 256, from the example in Table 3, we get the following plaintext password: *followthewhiterabbit*. This is the master password that was used for the user *victim_bob*

To summarize, if an user has the remember password option enabled, the master password is stored in a local database. The master password is encrypted with AES 256 CBC with the hash of the username as encryption key. Using this information, it is trivial to retrieve the master password when the remember password option is enabled.

5.2 Vault decryption

Once the master password is known, the user is very likely compromised. Using this technique, the attacker can login to the vault using the browser. However, when multifactor authentication is enabled, the attacker will not be able to access the credentials via the browser. Therefore, an attacker may be interested to find a way to retrieve the credentials stored in the local vault.

Recall that we have encountered the *LastPassData* table in the local database. This table contains multiple entries, one of which contains the type "accts". For clarification, an entry is displayed in Table 4.

The data field of this entry consists of two parts split by a semicolon. The first part indicates the number of iterations that is required for the key derivation function to create the key by which the accounts in the vault are encrypted. The second part is encoded in base64, which results in a blob of mostly binary data. In this binary file, several headers in ASCII format can be found, such as "ACCT" and "EQDN". We limit ourselves to the ACCT headers, which refer to the websites stored in the vault. Previous researchers have noted that not the whole vault is encrypted. This can be confirmed from the plaintext URLs from websites that can be observed the binary blob.

To discover the structure of the binary file, we searched the string "ACCT" in the code base. It appears that the function `parsemobile` in `lpfulllib.js` reads the data blob and extracts the fields. By setting debugging statements in this function, we can confirm that this function results in the data fields. The serialization function is quite complex; we, therefore, decided to search the binary file for the required fields. Using the debugging statements, the approximate order and position of the fields can be located in the file.

id	username_hash	type	data
231	f231c4103e569f8e4d6f3c0c...	accts	iterations=100100;TFBBVgAAAAIzMEFUVlI...

Table 4: Example content of an "accts" entry in the LastPassData table

For every website in the vault, the data is stored in one chunk. Every chunk starts of with a unix timestamp, followed by a sequence of unknown bytes. Further on in the chunk, we see a string of value "ACCT". After this string, there is another sequence of unknown bytes until a '!!' is encountered. This is an indicator of the next field, the field that contains the URL. At the start of the URL field, there are bytes of unknown meaning, followed by a sequence of ASCII encoded hexidecimal digits, which make up the string of the URL. The URL field is followed by a sequence of null bytes until the next delimiter is encountered. The encrypted username follows immediately after the URL field and continues until three null bytes are encountered. The encrypted password starts either direct after the username or after a '!' if this is present.

It should be clear to the reader that the reverse engineering effort of this file is time consuming to perform, error prone and hard to convey in this format. A more precise definition of the file format can be found in the parsing code of our tool. With the test cases that were used, our proposed format seems to be valid. At this point, we have obtainted the URL's, the encrypted username, and the encrypted password for every site in the vault. To decrypt the username and password, the vault key is required.

5.3 Vault key generation

The lastpass white paper states that the key that is used to encrypt and decrypt the fields in the vault is created by using 100100 iterations of PBKDF2-HMAC [13]. We can confirm the iteration count by observing the accounts entry in the local database, which is prepended with the iteration count. The PBKDF2-HMAC key is created using the username as salt and the master password as PBKDF password. The specific parameters are listed in Table 5. The key derivation results in the *vault key*; the key that is used to encrypt and decrypt the fields in the vault.

algorithm	SHA256
key length	32
salt	"victim_bob@protonmail.org"
password	"followthewhiterabbit"
iterations	100100

Table 5: Settings for PBKDF2-HMAC to create the vault key

The fields in the vault are encrypted with AES 256 CBC. Although, in this case, the IV is not separated by any delimeters. After looking at the source code of the extension, it seems that the first 16 bytes of the encrypted field is the IV; the rest of the field is the encrypted data. Using this knowledge, all credentials for the websites in the vault of the user can be decrypted. We have written a proof of concept tool that performs all steps described in the previous sections. The output of this tool is displayed in Figure 3.

5.4 Functional Requirements

The file system based attack has a two main requirements that must be met in order for this attack to be functional:

1. **Remember password needs to be enabled**: This is the most important requirement. When remember password is enabled, the master password is stored on disk and is near trivial to retrieve. This master password is required to unlock the locally stored vault.

2. **Offline mode needs to be enabled**: This requirement is important, but we would argue that this would almost always be met. By default, offline mode is enabled, which means that when the Lastpass servers become unavailable, it remains possible to access the account data.

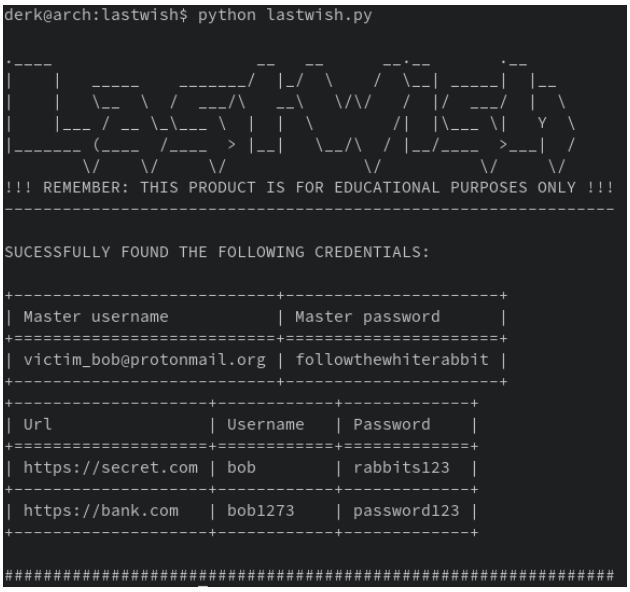

Figure 3: Output of our tool that abuses the remember password function to decrypt the locally stored vault.

> According to the Lastpass support page regarding vault management, the storage of a local encrypted vault is a design choice [14]. Offline mode can only be disabled in the MFA settings. As a result, MFA needs to be enabled to be able to disable offline mode.

Disabled MFA is not a requirement on this list. We have observed that with MFA, our method of the decryption of the vault remains valid.

6 Memory dump client-side attack

In this section, we will discuss the possibility of an attacker obtaining credentials from a memory dump of the Lastpass extension. The motive of this approach is the fact that usernames, passwords or cryptographic keys may be present in memory.

6.1 Obtaining the memory of the process

As with tabs in Chrome, every extension is assigned its own PID. In our case, we are interested in the PID of the LastPass extension. Using the key combination `Shift + Esc`, we obtain a list of chrome's processes where the PID of the extension can be found. It is important that the vault is unlocked when the memory is read, otherwise, the credentials and keys will not be present. The memory can be read using two approaches, either live while the process is active, or using a dump of the process. We have used the dump instead of the live memory as this was more convenient to search and analyse. The memory was dumped using WinDbg with the `.dump /f C:\path\to\dumpfile` command. Besides a dump, WinDbg can read while the memory of the process is still active. However, we have not used this feature.

6.2 Memory dump analysis

Once an attacker secures a dump of the password manager memory, they would be interested in either extracting credentials or keys. Because the memory dump contains the memory of the extension when the vault was unlocked, credentials may be found in the dump. A common method in Forensic Science to find plaintext strings, such as username or passwords, is using the unix *strings* command. This program can find strings in binary data that are above a certain length. When strings was run on the dump, the usernames and the names of the sites were found in plaintext; the passwords, however, are not present in plaintext in the dump.

We have noticed that the content of global JavaScript variables, as seen in the chrome developer tools, can be found back in the memory dump. Furthermore, the vault key can be found back in memory as well. This gives an attacker an opportunity to decrypt the locally stored accounts after obtaining a dump of the memory. In the next section, we will explore what approach can be taken to find this key in memory.

6.2.1 The search for the vault key

We have taken six memory dumps of the LastPass extension between both browser and operating restarts. An attacker would be interested to create a method to predict the position of the vault key in the memory dump of the extension. We have observed that the location of the key in memory is different for every dump. This is not unexpected as the variable will be allocated on the JavaScript heap. In order to predict the position of the variable in the JavaScript heap, if this is even possible, it would require in depth knowledge of the v8 JavaScript engine. Using the chrome developer tools, it is possible to inspect the objects on the JavaScript heap. However, we have not found a way to translate between the objects in the chrome developer tools to the objects in the memory dump.

We have searched for the vault key in order to find a predictable pattern. Once a pattern is found, this may be used to predict the vault key for different users and on different systems.

We know the vault key of bob in advance, as we can request this from the chrome developer tools:
`faea ad75 e058 e15b 3f83 d76f b14f a17c`
`90d4 4a43 b68c 91aa 81ef e786 a147 0ddd`

Using the *radare2* hexadecimal search function, we can find the vault key at the memory address 0x05870a63. The memory is displayed in Table 7 where the vault key is located on the fifth row. In the second and third dumps, the vault key could be found as well, however, it was located in at a different address. We noticed in each of the three dumps that the key was located near a certain string in memory. Because searching for this string provided a constant method to find the key for the user *victim_bob*, we tried to do the same for *victim_alice*. We observed that the key is no longer in the vicinity of this string, thus we were not able to find structures that would always be in proximity to the key. Lastly, we attempted to discover if enabling MFA would make a difference on the key in memory, but we did not observe a difference.

6.2.2 Find the vault key using regular expressions

After closer inspection, we have observed the following properties::

- The key is always 32 bytes long because 256 bits AES keys are used by LastPass.
- The key is always followed by 6 "unknown" bytes and 2 null bytes.
- There is always a 16 bit field before the key, with what we interpret to be the value of 0x20. This is present in different dumps, as well as in the two users this was tested on. We highly expect this to indicate the length of the next field, because 0x20 is 32 in decimal; this is the length of the vault key in bytes.

With this knowledge, we have created a regular expression in Python that matches the bytes in the memory dump of a potential key. The regular expression is as follows:

b"\x20\x00{3}[\x00-\xFF]{38}\x00\x00"

The preamble of the key starts with 0x20, followed by three null bytes. Next are the actual 32 bytes of the key, followed by 6 unknown bytes and two null bytes that indicate the end of the match. Using this expression on the first dump, it results in 18363 matches. The numbers of matches for this and the other dumps are displayed in Table 6. Note that the number of matches is high; although a brute force method is possible, it would be more elegant if this number can be reduced. We observe that there are many matches that contain a lot of zeroes and are very low in entropy, which is an unlikely property of a key to have. We therefore perform a very basic filter on the matches and remove the keys that contain zero bytes. The regex that is used for the raw match is very similar to the previous one:

b"\x20\x00{3}[\x01-\xFF]{38}\x00\x00"

This removes a significant amount of false positives; for the first dump, the number of matches decreased from 18363 to 224. However, there is a 12.5 % chance that a key will contain a zero byte, and will thus be

discarded. As this is a proof of concept, we will accept that this approach will not work for one eight of the keys. We now have a manageable number of possible keys that can be tried to decrypt the username and password fields in the vault. As a further improvement, we removed all key candidates that were entirely printable, this brought the number down to 90 possible keys. We are comfortable with this filter as the chance that a key is entirely printable is quite small. We expect that the filters can be further improved to result only in a handful or even a single key.

We have verified that in all these experiments, the actual key is present in the selection.

6.2.3 Finding the correct key

Once an attacker has a manageable set of possible keys, they could use the keys to decrypt one of the usernames in the vault. It would be reasonable to assume that the character set that is used for the username are all printable characters. The attacker could therefore iterate over every key to observe if the plaintext contains soley printable characters. If one of the keys results in a match, the attacker could use this candidate key to decrypt another one of the fields to verify that the candidate key is the vault key. When the attacker finds the correct key, the attacker will have full access to the accounts stored locally in the LastPass vault.

6.3 Functional requirements

This approach has limited requirements to correctly function. There are two important requirements that must be met:

1. **Offline access must be enabled**: This is enabled by default. If this would be disabled, it is still be possible to find the vault key in memory, but access to the offline encrypted vault would not be possible.

2. **The browser must be open and the extension unlocked**: Without this requirement satisfied, the attacker cannot create a dump of memory of the extension and can thus not search for the vault key in memory. Note that this is only required at the time of the dump; the analysis of the memory and the decryption of the credentials can all happen off-site.

A further minor requirement is that administrator rights may be needed in certain situations where the LastPass extension is run under a different user than the attacker is currently logged in as. If the attacker has access to the user it is targeting, no administrator rights are needed. Unlike the other client side attack that uses the remember password functionality to generate the vault key, this approach does not require the password to be remembered as the vault key will be extracted from the memory.

7 Implementation of the attacks

Both of our approaches are implemented in Python as proof of concepts. We created the tool called "LastWish" that is able to get the stored master password, generate the vault key and decrypt the vault. This tool requires the locally stored database; the attacker could thus copy the database to their machine and execute our tool. Regarding the memory attack, we have created a separate script that implements the described regular expressions to search for candidate keys in a memory dump. An attacker would first obtain a memory dump on the victims machine and could consecutively run the python script locally. The script of the memory based attack currently lacks the functionality to find the correct key from the selection of candidate keys, although this may be added in the future. The code base is available at the repository: `https://gitlab.os3.nl/dbarten/lastwish`.

8 Discussion

Approximately four and a half years after the study of Vigo et al. [6], the method for storing the remembered password has not been changed. Vigo et al. noted that, after disclosing the flaw, the remember password function will issue a pop up explaining the

Dump	Username	Matches	No null bytes	Not all printable
01	victim_bob	18363	224	90
02	victim_bob	19347	211	92
03	victim_bob	19008	205	88
04	victim_bob	15496	187	83
05	victim_alice	16512	190	90
06	victim_bob	16760	165	62

Table 6: The number of potential AES keys in the memory of the Lastpass extension. The "matches" column indicate the initial number of key candidates without filtering. The "no nullbytes" column indicates the key candidates that do not contain null bytes. The "not all printable" column indicate the number of key candidates that are not entirely printable. The "not all printable" key candidates are a subset of the "no null bytes" matches

0x05870a23	f12c	f8de	9f33	0000	d149	f8de	9f33	0000
0x05870a33	c18b	6384	cc6f	0000	091e	1136	526a	0000
0x05870a43	a125	f8de	9f33	0000	a125	f8de	9f33	0000
0x05870a53	c129	f8de	9f33	0000	0300	0000	2000	0000
0x05870a63	faea	ad75	e058	e15b	3f83	d76f	b14f	a17c
0x05870a73	90d4	4a43	b68c	91aa	81ef	e786	a147	0ddd
0x05870a83	0122	f8de	9f33	0000	0000	0000	e800	0000
0x05870a93	91ed	734e	975d	0000	2955	c648	0824	0000
0x05870aa3	311a	744e	975d	0000	915d	c648	0824	0000
0x05870ab3	6181	f859	dc49	0000	8126	f8de	9f33	0000

Table 7: The position of the vaultkey in the first dump

insecurity of the feature. As the remember password function is still present and unchanged, this leads us to believe that LastPass is not concerned about the security of this feature. In our opinion, the effort of LastPass to secure the remembered master password is minimal. It would be more accurate to view the encryption of master password as an attempt of obfuscation rather than an attempt of data security. A more secure approach could be to use native key storage instead, although we are not certain to what extend a browser extension is able access such resources.

Both the file system and memory based attacks rely on the fact that the encrypted vault is stored locally. As we have seen, when the password is remembered, this vault can be unlocked without much effort. Furthermore, the encryption key can be requested from the developer tools and memory. In our opinion, it would be safer for users to disable offline access by default. Currently, offline access can only be disabled if one or more forms of multi-factor authentication is enabled. We do not see a clear reason why offline access is linked to MFA and cannot be disabled independently; we recommend this to be a separate option. LastPass, by design, uses offline access as a method to allow users to login when the LastPass servers are unavailable. We would argue that offline access is a security weakness as this significantly increases the ability to steal credentials using client-side attacks.

The memory based attack was only performed in the Windows 10 operating system inside a virtual machine. Because Windows 10 was virtualized, a valid critique may be that the memory space can differ between a virtual machine and bare metal. We have not validated the method on bare metal and agree that this would be necessary to form conclusions about this attack on the Windows 10 operating system.

Furthermore, we have not performed this attack on different operating systems. We suspect that a similar approach will also be valid for different operating systems as long as a memory dump can be made of the extension. We have based our conclusions on two accounts and a total of 6 memory dumps of various states of the LastPass extension. We are of the opinion that more evidence would be required of different circumstances and users, thus we are only able to make limited conclusions about this attack.

We suspect that, with the memory based attack, the changing memory addresses are due to the v8 engine heap allocation internals. Furthermore, this may be caused by *Address Space Layout Randomization* (ASLR), although we believe this is unlikely. To clarify, ASLR is a security measure that allocates process and parts of processes, such as the stack and heap, to randomized locations. Thus the location of variables in a process will very likely be different every time the process is run. We do not suspect ASLR is the cause of the random memory locations because we have not inspected live memory but the dump of a single process instead. In the memory dump, all addresses are relative to the start of the dump. However, the position of the heap and stack of the process may be subjected to ASLR, despite the memory dump. Thus, to conclude, the cause of the random positions of variables in the memory dump is not clear to us.

The vault key that was used to unlock the vault in both the file system and memory based attack is available from the Chrome developer tools. The variable that contains the key can be requested from the console when the page `background.html` is requested. An attacker could therefore use the Chrome developer tools to get the key, instead of searching memory dump. It is very likely that LastPass is aware that the key can be requested from the console, as they mention a similar problem concerning the developer tools in their whitepaper [13]. We proceeded with using the memory dump to illustrate that this is a valid attack method. Because the memory based attack is in essence equal to extraction of the key from the developer tools, it is likely that LastPass is not concerned with this attack vector. If LastPass would mind these types of attacks, the key would have to be removed from memory, which, we expect, would result in significant architectural changes. To clarify, currently, the vault key is always present in memory, thus can be requested at any time to decrypt the vault. When the vault key is removed from memory, it needs to be generated from the master password every time the vault is decrypted. As a result, the user could be constantly prompted for their master password to generate the key; this may degrade the user experience significantly. We expect that LastPass is aware of the difficulty to defend against this type of client-side attacks and, therefore, do not include attack vector in their threat model.

Regarding the ethics of our research, both methods of attack are likely already known to LastPass and the public. In advance of our research, internet criminals were already able to perform an equivalent attack based on the file system using the metasploit module by Vigo et al. as this is readily available online. Hence, the method described in our research is already available to the public. Regarding the memory based attacks, these types of attacks are already well known but are not performed on the LastPass extension, as far as we are aware. The approach that we have described to gain the vault key from the memory can indeed be abused by criminals. However, we would like to remember the reader that the key can be read from the chrome developer tools as well. Thus, to conclude, our research has not significantly resulted in novel approaches to obtain passwords from the LastPass extension and, therefore, we expect the ethical impact to be limited.

Lastly, we wanted to discuss a few short points that we deem important to convey. We have not investigated what the impact of trusted and untrusted devices is on our approach. We have used the default installation of the extension and have not changed settings regarding trusted devices. We have not used a keylogger in our approach because we expected that this will not give new insights. Besides, a keylogger likely requires administrative rights, while our approaches are functional with user permissions. We have performed the attacks on the LastPass extension with merely two users that were created for this purpose. Because of this limited number and the time frame the accounts were created, it will not be possi-

ble to generalize to all users. For example, during our research, we have only encountered AES CBC while older accounts may still be using AES ECB. Furthermore, the method for extracting the username and password field from the locally stored database may not be valid for different users or vault contents.

9 Future work

As an extension of our research, it may be interesting to search for methods that will still be valid when offline mode is disabled. This would require research into the mechanisms that are used when this mode is disabled. As far as we are aware, when offline mode is disabled, the vault will be requested from the LastPass servers. We believe that this vault would be located somewhere in memory and can thus be read. Furthermore, it may be interesting to investigate whether the vault key can still be requested using the chrome developer tools.

10 Conclusion

For our restricted experimental setup, we were able to obtain credentials from the LastPass extension using both a file system based attack and a memory based attack. With the file system based approach, we are able to obtain the vault key when the remember password function is enabled, and thus could decrypt the encrypted credentials in the locally stored vault. Using the memory based attack, we were able to obtain the vault key from the memory of the LastPass extension. Using this key, we were able to decrypt the credentials in the vault using the same method as the file system based approach. We recommend LastPass to reconsider the offline access function, as that makes these two attacks possible.

References

[1] Dinei Florencio and Cormac Herley. "A large-scale study of web password habits". In: *Proceedings of the 16th international conference on World Wide Web*. ACM. 2007, pp. 657–666.

[2] Zhiwei Li et al. "The Emperor's New Password Manager: Security Analysis of Web-based Password Managers." In: *USENIX Security Symposium*. 2014, pp. 465–479.

[3] Mathias Karlsson. *How I made LastPass give me all your passwords*. URL: `https://labs.detectify.com/2016/07/27/how-i-made-lastpass-give-me-all-your-passwords/` (visited on 02/03/2019).

[4] David Silver et al. "Password Managers: Attacks and Defenses." In: *USENIX Security Symposium*. 2014, pp. 449–464.

[5] Joe Siegrist. *Security Update for the LastPass Extension*. URL: `https://blog.lastpass.com/2017/03/security-update-for-the-lastpass-extension.html/` (visited on 02/03/2019).

[6] A. Illera M. Vigo. *A look into LastPass*. 2014. URL: `https://www.martinvigo.com/a-look-into-lastpass/` (visited on 01/23/2019).

[7] Sherri Davidoff. "Cleartext passwords in linux memory". In: *Massachusetts institute of technology* (2008), pp. 1–13.

[8] Pasquale Stirparo et al. "In-memory credentials robbery on android phones". In: *Internet Security (WorldCIS), 2013 World Congress on*. IEEE. 2013, pp. 88–93.

[9] Dimitris Apostolopoulos et al. "Discovering authentication credentials in volatile memory of android mobile devices". In: *Conference on e-Business, e-Services and e-Society*. Springer. 2013, pp. 178–185.

[10] Kevin Breen. *LastPass - Recover memory resident account information*. URL: `https://github.com/kevthehermit/volatility_plugins` (visited on 02/03/2019).

[11] harmj0y. *A Case Study in Attacking KeePass*. 2016. URL: `http://www.harmj0y.net/blog/redteaming/a-case-study-in-attacking-keepass/` (visited on 01/08/2019).

[12] harmj0y. *KeeThief: A Case Study in Attacking KeePass part 2*. 2016. URL: http://www.harmj0y.net/blog/redteaming/keethief-a-case-study-in-attacking-keepass-part-2/ (visited on 01/08/2019).

[13] "Lastpass Technical Whitepaper". In: Lastpass. URL: https://assets.cdngetgo.com/1c/e4/e53646f14a91a7c9cb7dd7afbb61/lastpass-technical-whitepaper.pdf.

[14] *Manage your vault*. Lastpass. URL: https://support.logmeininc.com/lastpass/help/manage-your-vault-lp010015 (visited on 01/25/2019).