# A Security Analysis of Two Commercial Browser and Cloud Based Password Managers

Rui Zhao
University of Colorado Colorado Springs
Colorado Springs, CO 80918, USA
Email: rzhao@uccs.edu

Chuan Yue
University of Colorado Colorado Springs
Colorado Springs, CO 80918, USA
Email: cyue@uccs.edu

Kun Sun
George Mason University
Fairfax, VA 22030, USA
Email: ksun3@gmu.edu

***Abstract*—In this paper, we analyze the security of two very popular commercial password managers: LastPass and RoboForm. Both of them are *Browser and Cloud based Password Managers* (BCPMs), and both of them have millions of active users worldwide. We investigate the security design and implementation of these two BCPMs with the focus on their underlying cryptographic mechanisms. We identify several vulnerabilities that could be exploited by outsider and insider attackers to break the security of these two BCPMs.**

***Keywords*—Browser, password manager, security, cloud**

## I. Introduction

Text passwords still occupy the dominant position in online user authentication, and they cannot be replaced in the foreseeable future due to their security and especially their usability and deployability advantages [1], [2], [3]. Password security heavily relies on using strong passwords [4], [5] and protecting them from being guessed or stolen. However, strong passwords that are sufficiently long and random are often difficult to remember by users [4], [6], [7], [8]. Even if passwords are strong enough, they are still vulnerable to harvesting attacks such as phishing [9], [10]. These hard problems have been further aggravated by the facts that users have to create and manage more online passwords than before and they often have insecure practices such as sharing the same password across different websites [11] and writing down passwords [7].

Password manager is one of the most popular solutions that can potentially well address the aforementioned password security problems [2]. In general, password managers work by saving users' online passwords and later auto-filling the login forms on behalf of users. Therefore, a remarkable benefit brought by password managers is that users do not need to remember many passwords. This benefit is the main reason behind designing various password managers by many vendors and using them by millions of users.

All the major browser vendors have provided password manager as a built-in feature in their browsers (e.g., the top five most popular browsers: Internet Explorer, Firefox, Google Chrome, Safari, and Opera), and we analyzed the insecure design of browsers' built-in password managers in [12]. Third-party vendors have also provided many password managers. Popular commercial password managers often have two attractive properties: they are browser-based and cloud-based. We refer to such password managers as *Browser and Cloud based Password Managers* (BCPMs). They are browser-based in the sense they provide browser extension editions that can be seamlessly integrated into different Web browsers to achieve the same level of usability as browsers' built-in password managers. They are cloud-based in the sense they can store the saved websites passwords in the cloud storage servers and allow users to access the saved data from any place and at any time. This desired cloud-based usability property is not present or well supported in popular browsers, providing the opportunity for third-party vendors to gain a good share in the password manager market.

In this paper, we analyze the security of two very popular commercial BCPMs: LastPass [13] and RoboForm [14]. Both of them have millions of active users worldwide and are often ranked among the best password managers by media such as InformationWeek and PC Magazine. We investigate whether these two BCPMs are really secure and can properly protect users' online passwords. We make the following contributions: (1) define a threat model for analyzing the security of BCPMs, (2) investigate the design and implementation of these two BCPMs with the focus on their underlying cryptographic mechanisms, and (3) identify several vulnerabilities of these two BCPMs that could be exploited by outsider and insider attackers to obtain users' saved websites passwords.

## II. Background

LastPass is mainly designed and implemented as browser extensions for the top five most popular browsers [13]; we focus on its Firefox and Google Chrome browser extensions that share the same design. RoboForm has both stand-alone and browser extension editions [14]; we also focus on its Firefox and Google Chrome browser extensions that share the same design. In this paper, we use LastPass and RoboForm to refer to their Firefox and Google Chrome extensions, which are representative BCPMs that provide important usability benefits to users as highlighted in Section I.

Similar to other password managers, LastPass and RoboForm save users' websites login information (i.e., usernames and passwords for different websites), and later automatically fill the corresponding login forms on behalf of users. However, they interact with their cloud storage servers in different ways. LastPass always stores a user's websites login information to both the local machine and remote cloud storage servers. In contrast, RoboForm only allows a user to be in either the online mode or the offline mode. In the online mode, RoboForm stores a user's websites login information to its cloud storage servers; in the offline mode, it stores a user's websites login information to the user's local machine.

978-0-7695-5137-1/13 $26.00 © 2013 IEEE
DOI 10.1109/SocialCom.2013.70

Both BCPMs require a user to provide a BCPM username and BCPM password pair for authentication before allowing the user to access the saved data. Before saving a user's login information for any website, both BCPMs prompt a dialog box associated with the browser's address bar to obtain the user's confirmation. Once a user revisits the login webpage of a corresponding website, both BCPMs can auto-fill the login information on behalf of the user. In addition to using the extension's user interface, LastPass users can also log into the official LastPass website to manage their accounts.

To protect users' websites login information, both BCPMs take the approach of using a master password. The whole point of this approach is that the saved websites login information can only be decrypted and used by the user who provides the valid master password. In other words, even if attackers (including insincere LastPass or RoboForm employees) can obtain the saved data, they should not be able to feasibly decrypt and recover the original plaintext websites login information. The vendors of these two BCPMs claimed on their websites [13], [14] that they do not know users' master passwords, cannot resend or reset master passwords, and do not know users' login information for different websites.

## III. Security Analysis of LastPass and RoboForm

### A. Threat Model

We define the threat model for BCPMs from three perspectives: types of credentials, types of attackers, and types of attacks under consideration.

*1) Credentials:* We refer to a user's login information for different websites as `websites credentials`, which are the essential targets of attackers because the stolen websites credentials can be directly used to log into different websites to access and abuse a user's online accounts. We refer to a user's <BCPM username, BCPM password> pair as the `BCPM credential`, which allows a user to authenticate to the cloud storage servers of the corresponding BCPM through the LastPass or RoboForm browser extension. LastPass also allows a user to use the BCPM credential to log into its official website as described in Section II. The third type of credential is the `master password` that plays an important role in the security design of these two BCPMs. As will be soon analyzed, cracking the master password used in these two BCPMs can greatly facilitate the cracking of websites credentials.

*2) Attackers:* Two types of attackers may target those credentials: `outsider attackers` and `insider attackers`. Outsider attackers are unauthorized or illegitimate entities who initiate attacks from outside of the security perimeter of a BCPM vendor's system resources. They could be from amateur pranksters to organized criminals and even hostile governments. Outsider attackers may have the `server-side stealing capability`, i.e., intruding into the cloud storage servers of a BCPM vendor to steal the data saved for users. The attack happened on LastPass in 2011 [15] exemplifies such server-side stealing capability.

Outsider attackers may also have the `client-side stealing capability`, i.e., attacking users' machines to steal locally saved data. They may even have the `client-side computation capability`, i.e., temporarily running either benign or malicious programs on users' machines to perform some computations. For these two client-side capabilities, popular attacks such as drive-by-downloads [16], [17] are representative examples, in which attackers can install and run malware on a user's machine in a few seconds. We do not assume malware can persist on the victim's machine because anti-malware software may eventually detect and remove the malware. However, within that few seconds, the installed malware can either directly send back the stolen data for decrypting on attackers' machines, or, if necessary, decrypt the stolen data on the victim's machine and then send the results back to attackers.

Insider attackers are entities that are authorized to access a BCPM vendor's system resources but use them in a non-approved way. Examples of insider attackers could be insincere employees or former employees who can still access a BCPM vendor's system resources. Similar to outsider attackers, insider attackers may have the `server-side stealing capability` to steal the saved data. In addition, insider attackers may have the `server-side monitoring capability`, i.e., directly monitoring the communication between BCPMs and their cloud storage servers. Considering insider attackers in analyzing the security of BCPMs is of particular importance because although BCPM vendors store the encrypted data in their cloud storage servers, they should not be able to feasibly decrypt and recover any user's websites credentials and master password.

*3) Attacks Under Consideration:* We focus on the underlying cryptographic mechanisms of LastPass and RoboForm and mainly consider three types of attacks that could be performed to obtain credentials either from cloud storage servers or from users' local machines: `brute force attacks`, `local decryption attacks`, and `request monitoring attacks`.

Brute force attacks can be performed by both outsider and insider attackers to mainly crack a user's master password, from which other credentials can be further cracked. Note that we consider the effort of brute force attacks as the upper bound – attackers can definitely use different dictionaries to reduce their effort. Local decryption attacks aim to crack a user's websites credentials from the user's local machine without using brute force, and they can be performed by outsider attackers using drive-by-downloads and running malware on the victim's local machine. Request monitoring attacks aim to obtain a user's websites credentials by intercepting the requests sent from BCPMs to their cloud storage servers. Because BCPMs normally use the HTTPs protocol to secure their communication with cloud storage servers and meanwhile we do not assume malware can persist on a user's local machine, we mainly consider request monitoring attacks performed by insider attackers from the server-side.

### B. Security Analysis Methodology

We mainly investigated the two BCPMs on the Windows 7 platform. Both BCPMs are browser extensions written mainly in JavaScript, and their developers used different obfuscation techniques to make their JavaScript code difficult for other people to read and understand. Using Eclipse (www.eclipse.org) and JS Beautifier (jsbeautifier.org), we deobfuscate the JavaScript code of the two BCPMs for us to

analyze. Besides analyzing the source code, we use Mozilla's JavaScript debugger and Google Chrome's developer tools to help us understand the dynamic execution of the two BCPMs. To understand the communication between the two BCPMs and their cloud storage servers, we use the standalone edition HTTP Analyzer [18] to monitor and analyze all the incoming and outgoing traffic. To further confirm our understanding of the security design of the two BCPMs, we perform experiments and verify the related features such as storage, user authentication, and key derivation.

We estimate the effort of brute force attacks based on the computational power exemplified in a very popular cryptography textbook [19] authored by William Stallings. In the Table 2.2 (chapter 2, page 38, and 5th edition) of this textbook, Stallings used `two computer systems` with different computational power to estimate the brute force effort for searching cryptographic keys. The *first system* is more like a regular desktop computer, and it takes $\mathbf{10^{-6}}$ second to perform `a basic cryptographic operation`. The *second system* is more like a cluster of high performance servers with multi-core processors and GPUs, and it takes $\mathbf{10^{-12}}$ second to perform a basic cryptographic operation.

In our estimation, we consider either a DES (Data Encryption Standard) or an AES (Advanced Encryption Standard) decryption as a basic cryptographic operation as in [19]. Meanwhile, for simplicity but without loss of generality, we also consider either a SHA-1 or SHA-2 [20] hash operation as a basic cryptographic operation, although this is a conservative consideration because a hash operation is normally more efficient than a decryption operation. That means, in our estimation, the running time for each of these four basic cryptographic operations is $\mathbf{10^{-6}}$ second on the aforementioned *first system* and $\mathbf{10^{-12}}$ second on the aforementioned *second system*. We use this running time information in the following analysis and discussion of attackers' brute force effort.

### C. LastPass Security Design and Vulnerability Analysis

LastPass mainly uses JavaScript to support all of its functionalities including the cryptographic operations. It can also include an additional binary component to perform some cryptographic operations. If the binary component is not installed or not compatible with the system, cryptographic operations will be completely performed by JavaScript. LastPass always stores a user's websites credentials both locally to the user's machine and remotely to cloud storage servers.

A user only remembers a master password and a BCPM username. A *g_local_key* is derived from the master password and the BCPM username, and it will be used to encrypt the user's websites credentials. A *g_local_hash* is further derived from the master password and the g_local_key, and it will be used as the BCPM password. The <BCPM username, BCPM password> pair will be submitted to the cloud storage servers of LastPass for user authentication.

To perform both derivations, LastPass uses a variation of the deterministic password-based key derivation function *PBKDF2* specified in RFC 2898 [21]. The main variation is replacing the pseudorandom function recommended in the PBKDF2 specification [21] with the SHA-256 secure hashing function [20] to perform the underlying cryptographic operations. This replacement in LastPass was made probably for the purpose of ease of implementation, but it weakens the security of PBKDF2 because one major security improvement of PBKDF2 over its prior version PBKDF1 is using pseudorandom functions rather than hashing functions in the underlying cryptographic operations [21].

The PBKDF2 function used in LastPass (denoted as *PBKDF2_LP*) accepts four input parameters, in order: a *password*, a *salt*, an *iteration count*, and a *key length* value; it returns the derived key as the output. A SHA-256 operation is mainly performed for each iteration inside of the function; therefore, the iteration count parameter value corresponds to the total number of basic cryptographic operations performed in a *PBKDF2_LP* function call. LastPass derives *g_local_key* and *g_local_hash* by using *PBKDF2_LP* in Formula (1) and Formula (2), respectively. In Formula (1), a user's master password is used as the password parameter, the user's BCPM username is used as the salt, the iteration count is 500, and the derived *g_local_key* is 32 bytes. In Formula (2), *g_local_key* is used as the password parameter, the user's master password is used as the salt, the iteration count is one, and the derived *g_local_hash* is also 32 bytes.

$$g\_local\_key = PBKDF2\_LP(master\ password, BCPM\ username, 500, 32) \quad (1)$$

$$g\_local\_hash = PBKDF2\_LP(g\_local\_key, master\ password, 1, 32) \quad (2)$$

We now reveal the vulnerabilities in the security design of LastPass and discuss three types of potential attacks: *outsider attackers' local decryption attacks*, *outsider attackers' brute force attacks*, and *insider attackers' brute force attacks*. We analyze how a user's master password can be cracked. With the cracked master password, attackers can directly derive the *g_local_key* to completely decrypt all the websites credentials of the user, and can further derive the BCPM password (i.e., *g_local_hash*) of the user.

*1) Outsider Attackers' Local Decryption Attacks:* The vulnerability lies in the *insecure design of the master password remembering mechanism in LastPass*. LastPass can even remember a user's master password (with the BCPM username) into a local SQLite [22] database table *LastPassSavedLogins2*, allowing the user to be automatically authenticated whenever LastPass is used again. Whether and how LastPass protects the master password before saving it into the database table depends on the configuration of the user's machine. There are three possible cases: (1) if LastPass includes an aforementioned binary component and the TPM (Trusted Platform Module) of the machine is available, the *protect_data()* function of the binary component will use the Windows API function *CryptProtectData()* with the TPM support to encrypt the master password; (2) if the binary component exists but the TPM of the machine is not available, the *protect_data()* function will use *CryptProtectData()* without the TPM support to encrypt the master password; and (3) if the binary component does not exist, LastPass will not encrypt the master password at all.

A locally saved master password, no matter encrypted or not, is vulnerable to local decryption attacks that can be performed by outsider attackers with the client-side stealing capability and/or the client-side computation capability (Sec-

tion III-A2). In the cases (1) and (2) where the *protect_data()* function of the binary component is used in the encryption, outsider attackers can call the corresponding *unprotect_data()* function of the binary component on the victim's machine to decrypt the master password. In other words, attackers need to have both the client-side stealing capability and the client-side computation capability. The *unprotect_data()* function will use the corresponding Windows API function *CryptUnprotectData()* either with or without the TPM support (based on the configuration of the user's machine) to perform the decryption. In the case (3) where no encryption is applied, outsider attackers with the client-side stealing capability can directly steal the saved plaintext master password.

In all the three cases, outsider attackers can directly steal the plaintext BCPM username from the *LastPassSavedLogins2* database table. Therefore, using Formula (1) and Formula (2), outsider attackers can derive *g_local_key* and *g_local_hash* to completely recover all the plaintext websites credentials of a user. We performed experiments and validated the effectiveness of such local decryption attacks. The time effort of such attacks is very low – within one second, the entire decryption process can be completed and all the plaintext websites credentials of a user can be accurately obtained by outsider attackers.

*2) Outsider Attackers' Brute Force Attacks:* Even if a master password is not saved by LastPass into the *LastPassSavedLogins2* database table on a user's local computer, it is still vulnerable to brute force attacks performed by outsider attackers. The vulnerability lies in the *insecure design of the local user authentication mechanism and the insecure application of the PBKDF2 function in LastPass*.

To locally authenticate a user and make the user's websites credentials accessible when the network connection is not available, LastPass encrypts a hard-coded string "lastpass rocks" using AES and writes the ciphertext into another local SQLite [22] database table *LastPassData*, in which the encrypted websites credentials are also saved. The key used in this AES encryption operation is the same key (i.e., *g_local_key*) used for encrypting a user's websites credentials. Therefore, in a local user authentication, if the key derived from Formula (1) based on the BCPM username and the master password provided by a user can decrypt the ciphertext for "lastpass rocks" back to the correct plaintext, the authentication will be successful and LastPass will further decrypt the websites credentials for the user.

Outsider attackers with the client-side stealing capability (Section III-A2) can perform brute force attacks using the following steps after stealing the BCPM username and the ciphertext for "lastpass rocks". First, an attacker derives *g_local_key* (Formula (1)) by trying one possible master password together with the stolen BCPM username. Second, the attacker tries to decrypt the ciphertext for "lastpass rocks" using AES with the derived *g_local_key* as the decryption key. Third, if the decrypted result is "lastpass rocks", the brute force attack is successful and the attacker obtains the user's real master password; otherwise, the attacker repeats above steps with another possible master password. Each master password try consists of 501 (500 iterations in Formula (1) plus one AES decryption) basic cryptographic operations, thus taking $501*\mathbf{10^{-6}}$ seconds and $501*\mathbf{10^{-12}}$ seconds, respectively, on the two systems referred in Section III-B.

The effectiveness of such brute force attacks also depends on the size of the master password space, which is determined by the length of the master password and the number of possibilities for each master password character. If each master password character can be an upper case letter, a lower case letter, or a decimal digit, then it could be one of the 62 (26+26+10) possibilities. Based on this number, we list different master password lengths and their corresponding space sizes in the first column and the second column of Table I, respectively. The third and fourth columns of Table I list the outsider attackers' average brute force attack effort (i.e., overall effort divided by two) with one try's running time at $501*\mathbf{10^{-6}}$ seconds and $501*\mathbf{10^{-12}}$ seconds, respectively. For example, on average, outsider attackers can crack an 8-character master password in about 1734.3 years and 15.2 hours, respectively, on the aforementioned two systems.

TABLE I. THE AVERAGE BRUTE FORCE ATTACK EFFORT ON THE MASTER PASSWORD FOR LASTPASS.

| Master password length | Master password space size | Outsider attackers' brute force attack effort with one try's running time at: | | Insider attackers' brute force attack effort with one try's running time at: | |
|---|---|---|---|---|---|
| | | $501*\mathbf{10^{-6}}$ seconds | $501*\mathbf{10^{-12}}$ seconds | $2*\mathbf{10^{-6}}$ seconds | $2*\mathbf{10^{-12}}$ seconds |
| 5 | $\mathbf{62^5}$ | 2.7 days | 0.2 seconds | 15.3 minutes | $9*\mathbf{10^{-4}}$ seconds |
| 6 | $\mathbf{62^6}$ | 164.7 days | 14.3 seconds | 15.8 hours | 0.06 seconds |
| 7 | $\mathbf{62^7}$ | 28 years | 14.7 minutes | 40.8 days | 3.5 seconds |
| 8 | $\mathbf{62^8}$ | 1734.3 years | 15.2 hours | 6.9 years | 3.7 minutes |
| 9 | $\mathbf{62^9}$ | $1.1*\mathbf{10^5}$ years | 39.3 days | 430 years | 3.8 hours |
| 10 | $\mathbf{62^{10}}$ | $6.5*\mathbf{10^6}$ years | 6.7 years | $2.7*\mathbf{10^4}$ years | 9.7 days |

*3) Insider Attackers' Brute Force Attacks:* Insider attackers with the server-side monitoring capability (Section III-A2) can perform brute force attacks on a user's master password. The vulnerability lies in the *insecure association of the master password with authenticators in LastPass*. The brute force attacks can be performed in two different ways (note that outsider attackers with the harvested BCPM credential or a double-hashed value, e.g., harvested by phishing with a spoofed LastPass website, can perform the same attacks). One way is to intercept the BCPM credential (i.e., the <BCPM username, *g_local_hash*> pair) and then perform the same brute force attacks as we just described for outsider attackers. Therefore, the brute force attack effort is the same as that listed in the third and fourth columns of Table I.

The second way is to intercept the double-hashed value sent to the official website of LastPass. When a user logs into the official website of LastPass using a browser, a SHA-256 double-hashed value generated from the BCPM username and the master password is also sent to the server. Brute force attacks against the master password can be more efficiently performed by insider attackers with the intercepted double-hashed value. An insider attacker only needs to calculate the double-hashed value (i.e., two basic cryptographic operations) from the BCPM username and a possible master password. If the calculated double-hashed value matches the intercepted one, the brute force attack is successful and the attacker recovers the user's master password; otherwise, the attacker repeats the calculation on another possible master password. Each master password try takes $2*\mathbf{10^{-6}}$ seconds and $2*\mathbf{10^{-12}}$ seconds, respectively, on the two systems referred in Section III-B; the fifth and sixth columns of Table I list the average brute force attack effort of insider attackers.

### D. RoboForm Security Design and Vulnerability Analysis

RoboForm is implemented in pure JavaScript and it has an online mode and an offline mode. In the offline mode, RoboForm stores a user's websites credentials to the user's local machine. In the online mode, RoboForm uploads a user's websites credentials to its remote cloud storage servers through the HTTPS communication. In the offline mode, RoboForm also uses a variation of the *PBKDF2* [21]. The main variation is replacing the pseudorandom function recommended in the PBKDF2 specification [21] with the SHA-1 secure hashing function [20] to perform the underlying cryptographic operations. Similar to that of LastPass, such a replacement weakens the security of PBKDF2. Meanwhile, using SHA-1 rather than SHA-2 [20] further weakens the security.

The PBKDF2 function used in RoboForm (denoted as *PBKDF2_RF*) has the same interface as in *PBKDF2_LP*, i.e., it accepts four input parameters and returns the derived key as the output. The difference is that two SHA-1 operations are mainly performed for each iteration inside of the *PBKDF2_RF* function; therefore, the iteration count parameter value corresponds to one half of the total number of basic cryptographic operations performed in a *PBKDF2_RF* function call. RoboForm derives a key by using *PBKDF2_RF* in Formula (3). A user's master password is used as the password parameter, a random number is used as the salt, the iteration count is 1000, and the derived key is 34 bytes.

$$key = PBKDF2\_RF(master\ password, random\ number, 1000, 34) \quad (3)$$

We now reveal the vulnerabilities in the security design of RoboForm and discuss three types of potential attacks: *outsider attackers' local decoding attacks*, *outsider attackers' brute force attacks*, and *insider attackers' request monitoring attacks*. The first two types of attacks are related to the offline mode of RoboForm. The third type of attacks are related to the online mode of RoboForm.

*1) Outsider Attackers' Local Decoding Attacks:* The vulnerability lies in the *zero protection to local storage when a master password is not used in RoboForm*. In the offline mode, RoboForm saves each website credential into a separate *.rfp* file. Each *.rfp* file is organized into three parts: a *header*, a *flag*, and a *data block*. The header is always a string concatenated from a hard-coded string "URL3:ver3" and the encoded website login URL. The formats of the other two parts depend on whether a master password has been used. In the case when a master password is not used, the flag will be a hard-coded string "@PROTECTED@" and the data block will be the encoded format of a user's website credential. In other words, a user's website credential is not encrypted at all, it is simply encoded without using any cryptographic key. The encoding and decoding schemes are implemented in the RoboForm *RfGarbleString()* and *RfUngarbleStringW()* JavaScript functions, respectively.

Therefore, outsider attackers with the client-side stealing capability (Section III-A2) can simply steal the *.rfp* files of those RoboForm users who do not use a master password. With the stolen *.rfp* files, outsider attackers can run the decoding function *RfUngarbleStringW()* on any computer to completely recover a user's websites credentials. Note that local decoding attacks can be regarded as the simplest and most special forms of local decryption attacks in which no decryption keys are needed to recover the plaintext. We performed experiments and validated the effectiveness of such local decoding (or decryption) attacks. The time effort of such attacks is very low – within one second, the entire decoding process can be completed and all the plaintext websites credentials of a user can be accurately obtained by outsider attackers.

*2) Outsider Attackers' Brute Force Attacks:* In the offline mode and if a master password has been used, outsider attackers with the client-side stealing capability (Section III-A2) can still perform brute force attacks against a user's master password. With the cracked master password, attackers can further obtain all the websites credentials of the user. The vulnerability lies in the *weak protection to the local storage when a master password is used in RoboForm*. In more details, the brute force attacks can be performed in two different ways: based on the *.rfp* files, or based on the *smpenc.rfo* file.

*(a) Based on the .rfp files:* In the case when a master password is used, in each *.rfp* file, the flag will be a hard-coded string "+PROTECTED-2+", and the data block will consist of an 8-byte salt, a 2-byte password verification code, a 10-byte integrity checksum, and a ciphertext. The salt is a random number used as the second input parameter to the *PBKDF2_RF* (Formula (3)) function. The first 32 bytes of the derived key will be used in the AES encryption to convert a website credential into the ciphertext. The password verification code comes from the last two bytes of the derived key, and it is used to verify the correctness of a user's master password in the offline mode. The integrity checksum is calculated from the HMAC (Keyed-Hashing for Message Authentication, RFC 2104) function on the website credential using the second 16 bytes of the derived key, and it is used to verify the integrity of the data saved in the *.rfp* file.

Therefore, with the stolen *.rfp* files, outsider attackers can first derive a key from a possible master password using the *PBKDF2_RF* function with 1000 iterations. They can then compare the calculated password verification code with the one saved in a *.rfp* file. If a comparison is successful, they can further decrypt the ciphertext and verify the calculated integrity checksum against the one saved in the *.rfp* file. If this final verification is successful, the brute force attack is successful; otherwise, if any mismatch happens, attackers can simply try another possible master password. Each master password try consists of 2001 (1000 iterations in Formula (3) with two SHA-1 operations in each iteration plus one AES decryption) basic cryptographic operations. Therefore, each master password try takes $2001*\mathbf{10^{-6}}$ seconds and $2001*\mathbf{10^{-12}}$ seconds, respectively, on the two systems referred in Section III-B; the third and fourth columns of Table II list the corresponding average brute force attack effort of outsider attackers.

*(b) Based on the smpenc.rfo file:* In the case when a master password is used in the offline mode, a *smpenc.rfo* file is also created by RoboForm. A user's master password concatenated with a hard-coded string "MASTER PASSWORD FILE" will be encrypted using a single DES (1-DES) operation with a 56-bit key. The key itself is derived from the user's master password using a *RFGenerateKey()* JavaScript function, which simply takes the first 8 bytes of the master password and performs a naive transformation without involving any additional data. The ciphertext is saved into the *smpenc.rfo* file.

TABLE II. THE AVERAGE BRUTE FORCE ATTACK EFFORT ON THE MASTER PASSWORD FOR ROBOFORM.

| Master password length | Master password space size | Based on the *.rfp* files, outsider attackers' brute force attack effort with one try's running time at: | | Based on the *smpenc.rfo* file, outsider attacker's brute force attack effort with one try's running time at: | |
|---|---|---|---|---|---|
| | | $2001*10^{-6}$ seconds | $2001*10^{-12}$ seconds | $1*10^{-6}$ seconds | $1*10^{-12}$ seconds |
| 5 | $62^5$ | 10.6 days | 0.9 seconds | 7.7 minutes | $4.6*10^{-4}$ seconds |
| 6 | $62^6$ | 1.8 years | 10 minutes | 7.9 hours | $2.9*10^{-2}$ seconds |
| 7 | $62^7$ | 110 years | 1.0 hours | 20.4 days | 1.8 seconds |
| 8 | $62^8$ | $7.0*10^3$ years | 2.6 days | 3.5 years | 1.8 minutes |
| 9 | $62^9$ | $4.3*10^5$ years | 157.0 days | 215 years | 1.9 hours |
| 10 | $62^{10}$ | $2.7*10^7$ years | 26.7 years | $1.3*10^4$ years | 4.9 days |

RoboForm uses this *smpenc.rfo* file to authenticate an offline user. However, this user authentication mechanism makes a user's master password very vulnerable to brute force attacks performed by outsider attackers with the client-side stealing capability (Section III-A2). Once stealing a user's *smpenc.rfo* file, outsider attackers first derive a decryption key using the *RFGenerateKey()* function with a possible master password, then decrypt (using 1-DES) the ciphertext stored in the *smpenc.rfo* file, and finally verify whether the decrypted result is the concatenation of the tried master password and the hard-coded string "MASTER PASSWORD FILE". If the verification is successful, the brute force attack is successful; otherwise, attackers can simply try another possible master password. Each master password try consists of one basic cryptographic operation, which is the 1-DES decryption because the overhead of the naive transformation in the *RFGenerateKey()* function can be ignored. Therefore, each master password try takes $1*10^{-6}$ seconds and $1*10^{-12}$ seconds, respectively, on the two systems referred in Section III-B; the fifth and sixth columns of Table II list the corresponding average brute force attack effort of outsider attackers.

Comparing to the brute force attacks based on the *.rfp* files, brute force attacks based on the *smpenc.rfo* file are more efficient. With the same client-side stealing capability (Section III-A2) requirement in both types of attacks, it is reasonable to believe that attackers would choose to take the efficient approach of using the stolen *smpenc.rfo* file.

*3) Insider Attackers' Server-side Request Monitoring Attacks:* In the online mode, a user's credentials including the master password, BCPM credential, and websites credentials will be sent to the cloud storage servers of RoboForm through the HTTPS communication. The vulnerability lies in the *zero protection to the data received by the insiders of RoboForm*.

As we verified through source code inspection and traffic analysis, RoboForm does not encrypt any of those information – it simply transmits the plaintext of those information through the HTTPS communication. For example, when a user registers a RoboForm account, the BCPM credential is sent to the cloud storage servers in plaintext; when a user remembers a website credential using RoboForm, the website credential is sent to the cloud storage servers in plaintext; when a user sets the master password, the master password is sent to the cloud storage servers in plaintext; when a user asks RoboForm to auto-fill a website login form, the cloud storage servers will send back the website credential in plaintext.

Therefore, although HTTPS encrypts the client-server communication and protects against the man-in-the-middle attacks, insider attackers with the server-side monitoring capability (Section III-A2) can directly and completely obtain all the credentials of a user – they simply need to monitor the incoming HTTPS requests and wait for their decryption at the server-side. This is a severe vulnerability because insiders (BCPM vendors) should not be able to feasibly decrypt and recover any user's websites credentials and master password as we highlighted in the threat model for BCPMs.

## IV. CONCLUSION

In this paper, we analyzed the security design of two very popular commercial BCPMs: LastPass and RoboForm. We identified several vulnerabilities in both BCPMs and analyzed how outsider and insider attackers can exploit those vulnerabilities to perform different attacks.

## REFERENCES


[1] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Proc. of IEEE S&P Symposium*, 2012.

[2] C. Herley and P. C. van Oorschot, "A research agenda acknowledging the persistence of passwords," *IEEE Security & Privacy*, vol. 10, no. 1, pp. 28–36, 2012.

[3] C. Herley, P. C. van Oorschot, and A. S. Patrick, "Passwords: If we're so smart, why are we still using them?" in *Proc. of FC*, 2009.

[4] D. C. Feldmeier and P. R. Karn, "Unix password security – ten years later," in *Proc. of CRYPTO*, 1989.

[5] R. Morris and K. Thompson, "Password security: a case history," *Commun. ACM*, vol. 22, no. 11, pp. 594–597, 1979.

[6] A. Adams and M. A. Sasse, "Users are not the enemy," *Commun. ACM*, vol. 42, no. 12, pp. 40–46, 1999.

[7] S. Komanduri, R. Shay, P. G. Kelley, M. L. Mazurek, L. Bauer, N. Christin, L. F. Cranor, and S. Egelman, "Of passwords and people: Measuring the effect of password-composition policies," in *Proc. of CHI*, 2011.

[8] J. Yan, A. Blackwell, R. Anderson, and A. Grant, "Password memorability and security: Empirical results," *IEEE Security and Privacy*, vol. 2, no. 5, pp. 25–31, 2004.

[9] M. Jakobsson and S. Myers, *Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft*. Wiley-Interscience, ISBN 0-471-78245-9, 2006.

[10] Rachna Dhamija and J.D.Tygar and Marti Hearst, "Why phishing works," in *Proc. of CHI*, 2006.

[11] D. Florêncio and C. Herley, "A large-scale study of web password habits," in *Proc. of WWW*, 2007.

[12] R. Zhao and C. Yue, "All Your Browser-saved Passwords Could Belong to Us: a Security Analysis and a Cloud-based New Design," in *Proc. of CODASPY*, 2013.

[13] "LastPass Password Manager." https://lastpass.com/.

[14] "RoboForm Password Manager." http://www.roboform.com/.

[15] "LastPass, Online Password Manager, May Have Been Hacked," http://www.pcworld.com/article/227223/LastPass_Online_Password_Manager_May_Have_Been_Hacked.html.

[16] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose, "All your iframes point to us," in *Proc. of USENIX Security Symposium*, 2008.

[17] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. T. King, "Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities." in *Proc. of NDSS*, 2006.

[18] "HTTP Analyzer," http://www.ieinspector.com/httpanalyzer/index.html.

[19] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 5th ed. Prentice Hall Press, 2010.

[20] "NIST: Secure Hashing," http://csrc.nist.gov/groups/ST/toolkit/secure_hashing.html.

[21] B. Kaliski, "RFC 2898," 1999, http://www.ietf.org/rfc/rfc2898.txt.

[22] "SQLite Home Page," http://www.sqlite.org.