Toward A Secure and Usable Cloud-based Password Manager for Web Browsers

Rui Zhao and Chuan Yue

Department of Computer Science, University of Colorado Colorado Springs, USA, 80918
E-mail:rzhao@uccs.edu and cyue@uccs.edu

Abstract

Web users are confronted with the daunting challenges of creating, remembering, and using more and more strong passwords than ever before in order to protect their valuable assets on different websites. Password manager, particularly Browser-based Password Manager (BPM), is one of the most popular approaches designed to address these challenges by saving users' passwords and later automatically filling the login forms on behalf of users. Fortunately, all the five most popular Web browsers have provided password managers as a useful built-in feature. In this paper, we uncover the vulnerabilities of existing BPMs and analyze how they can be exploited by attackers to crack users' saved passwords. Moreover, we propose a novel Cloud-based Storage-Free BPM (CSF-BPM) design to achieve a high level of security with the desired confidentiality, integrity, and availability properties. We have implemented a CSF-BPM system into Firefox and evaluated its correctness, performance, and usability. Our evaluation results and analysis demonstrate that CSF-BPM can be efficiently and conveniently used. We believe CSF-BPM is a rational design that can also be integrated into other popular browsers to make the online experience of Web users more secure, convenient, and enjoyable.

Keywords: Web Browser, Password Manager, User Authentication, Security, Cloud, Storage

1. Introduction

Text-based passwords still occupy the dominant position in online user authentication [1, 2, 3]. They protect online accounts with valuable assets, and thus have been continuously targeted by various cracking and harvesting attacks. Password security heavily depends on creating strong passwords and protecting them from being stolen. However, researchers have demonstrated that strong passwords that are sufficiently long, random, and hard to crack by attackers are often difficult to remember by users [4, 5, 6, 7, 8]. Meanwhile, no matter how strong they are, online passwords are also vulnerable to harvesting attacks such as phishing [9, 10, 11]. These hard problems have been further aggravated by the fact that Web users have more online accounts than ever before, and they are forced to create and remember more and more usernames and passwords probably using insecure practices such as sharing passwords across websites [12, 13].

Password manager, particularly Browser-based Password Manager (BPM) is one of the most popular approaches that can potentially well address the online user authentication and password management problems. Browser integration enables BPMs to easily save users' login information including usernames and passwords into a database, and later automatically fill the login forms on behalf of users. Therefore, users do not need to remember a large number of strong passwords; meanwhile, BPMs will only fill the passwords on the login forms of the corresponding websites and thus can potentially protect against phishing attacks. Fortunately, mainly to support the password autofill and management capability, all the five most popular browsers Internet Explorer, Firefox, Google Chrome, Safari, and Opera have provided password managers as a useful built-in feature.

In this paper, we uncover the vulnerabilities of existing BPMs and analyze how they can be exploited by attackers to crack users' saved passwords. Moreover, we propose a novel Cloud-based Storage-Free BPM (CSF-BPM) design to achieve a high level of security with the desired confidentiality, integrity, and availability properties. CSF-BPM is cloud-based storage-free in the sense that the protected data will be completely stored in the cloud – nothing needs to be stored on a user's computer. We want to move the storage into the cloud for two main reasons. One is that in the long run trustworthy storage services in the cloud [14, 15, 16, 17, 18, 19] can better protect a regular user's data than local computers (which may not be timely and properly patched) do, especially if a storage service uses secret sharing schemes such as the (k, n) threshold scheme [20] to only save pieces of the encrypted data to different cloud vendors [14]. The other reason is that the stored data can be easily accessible to the user across different OS accounts on the same computer and across computers at different locations at anytime.

We have implemented a CSF-BPM system and seamlessly integrated it into the Firefox Web browser. We have evaluated the correctness, performance, and usability of this system. We believe CSF-BPM is a rational design that can also be integrated into other popular browsers to make the online experience of Web users more secure, convenient, and enjoyable. We have followed standard responsible disclosure practices and reported those vulnerabilities to the respective browser vendors. Our vulnerability verification tools and the CSF-BPM system can be demonstrated and be shared with responsible researchers.

We provide four main contributions in this paper. First, we compare the BPMs of the five most popular browsers and identify the inconsistencies in their functionality and interface designs (Section 2). Second, we uncover the security vulnerabilities of the five BPMs and analyze how they can be exploited by attackers to crack users' saved passwords (Section 3). Third, we propose a novel CSF-BPM design to achieve a high level of security (Section 4). Finally, we present an implementation (Section 5) and evaluation (Section 6) of the Firefox version CSF-BPM system, and discuss its limitations (Section 7).

2. Related Work and Background

In this section, we briefly review the related password and password manager research, and provide the background information on the BPMs of the five most popular browsers.

2.1. Related Work

Morris and Thompson pointed out long ago in 1979 that weak passwords suffer from brute-force and dictionary attacks [7]. Later, Feldmeier and Karn further emphasized that increasing password entropy is critical to improving password security [5]. However, strong passwords that are sufficiently long, random, and hard to crack by attackers are often difficult to remember by users due to human memory limitations. Adams and Sasse discussed password memorability and other usability issues and emphasized the importance of user-centered design in security mechanisms [4]. Yan et al. [8] analyzed that strong password requirements often run contrary to the properties of human memory, and highlighted the challenges in choosing passwords that are both strong and mnemonic. Recently, Florêncio and Herley performed a large-scale study of Web password habits and demonstrated the severity of the security problems such as sharing passwords across websites and using weak passwords [12]. A large-scale user study recently performed by Komanduri et al. demonstrated that many Web users write down or otherwise store their passwords, and especially those higher-entropy passwords [6].

To help Web users better manage their online accounts and enhance their password security, researchers and vendors have provided a number of solutions such as password managers [21, 22, 23], Web Single Sign-On (SSO) systems [24, 25, 26, 27], graphical passwords [28, 29, 30], and password hashing systems [31, 32, 33]. As analyzed in Section 1, password managers especially BPMs have the great potential to well address the challenges of using many strong passwords and protecting against phishing attacks. The insecurity of third-party commercial password managers such as LastPass [34] and RoboForm [23] are analyzed by Zhao et al. in [35]. Web Wallet [21] is an anti-phishing solution and is also a password manager that can help users fill login forms using stored information; however, as pointed out by the authors, users have a strong tendency to use traditional Web forms for typing sensitive information instead of using a special browser sidebar user interface. In addition, Web Wallet is not cloud-based. In terms of Web SSO systems, their security vulnerabilities such as insecure HTTP referrals and implementations are analyzed in [24, 36, 37], their business model limitations such as insufficient adoption incentives are analyzed by Sun et al. in [25], and their vulnerabilities to phishing attacks against the identity provider (such as Google and Facebook) accounts are highlighted by Yue in [38]. Security limitations of graphical passwords are analyzed in [28, 29, 30]. Security and usability limitations of password hashing systems are analyzed in [39, 31]. We do not advocate against any of these other approaches. We simply focus on the BPM security in this paper.

2.2. Password Manager Feature of Browsers

Table 1 lists the basic information on the BPM feature of the recent versions of the five most popular Web browsers. The second column of the table provides the sequence of menu items that a user must click in order to finally access the BPM feature configuration interface. We can see that the BPM feature configuration locations are very different among browsers. Indeed, the feature configuration interfaces shown on those locations are also very different among browsers in terms of the configuration options and functions. The third column shows that the BPM feature is enabled by default in four browsers but not in Safari. The fourth column shows that only Firefox employs a master password mechanism, which is, however, not enabled by default and users may not be aware of its importance. Note that Opera employed a weak master password mechanism in its early versions such as version 12.02 [40]. The fifth column shows that Firefox, Google Chrome, and Opera provide a password synchronization mechanism that can allow users to access the saved passwords across different computers.

Table 1: Basic information of BPMs in five most popular browsers.

Browser	Configuration Location	Enabled by Default	Master Password	Password Sync.
Internet Explorer (11.0)	Internet options → Content → AutoComplete Settings → User names and passwords on forms	Yes	No	No
Firefox (27.0)	Options → Security → Passwords	Yes	Yes	Yes
Google Chrome (33.0)	Settings → Show advanced settings... → Passwords and forms	Yes	No	Yes
Safari (5.1.7)	Preferences → AutoFill → User names and passwords	No	No	No
Opera (20.0)	Settings → Privacy & security → Passwords	Yes	No	Yes

In terms of the dynamic behavior, the interfaces for triggering the remembering and autofill of passwords are inconsistent among browsers. For one example, all the browsers display a dialog box to ask a user whether the entered password for the current website should be remembered. The dialog boxes displayed by Firefox, Google Chrome, and Opera are associated with the address bar, thus technically hard to be spoofed. For another example, Internet Explorer, Firefox, and Opera require a user action before auto-filling the password value on a website; however, Google Chrome and Safari autofill the username and password values once a user visits a login webpage, providing more opportunities for malicious JavaScript to manipulate the login form and information.

Overall, the BPM interface design is very inconsistent among these five browsers. The security implications of these interface inconsistencies will be investigated in our future work. In this paper, we simply focus on the BPM security design itself.

3. Vulnerability Analysis

In this section, we first define the threat model and assumptions that we consider throughout this paper. We then use an analogy to justify the essential problem of existing BPMs. Finally, we provide a detailed vulnerability analysis regarding without and with a master password mechanism.

3.1. Threat Model and Assumptions

"Where a threat intersects with a vulnerability, risk is present [41]." For Browser-based Password Managers (BPMs), the threat sources are attackers who want to steal the sensitive login information stored by BPMs.

3.1.1. Threat Model

Our basic threat model is that attackers can temporarily install malware such as Trojan horses and bots on a user's computer using popular attacks such as drive-by downloads [42, 43, 44, 45, 46]. The installed malware can then steal the login information stored by BPMs. For example, Stone-Gross et al. inferred that 38% of the credentials stolen by the Torpig bot were obtained from the password managers of browsers, rather than by intercepting an actual login session [13]. Note that the malware can run at the system-level or at the application-level, and can even be malicious browser extensions [47]. Indeed, if the occurrences of such threats are rare or do not have high impacts, BPMs would not bother to encrypt their stored passwords in the first place. Therefore, our focus will be on investigating the vulnerabilities of BPMs that could be exploited by potential threat sources to easily decrypt the passwords stored by BPMs.

3.1.2. Assumptions

We assume that it is very difficult for the installed malware to further compromise the operating system to directly identify cryptographic keys from a computer's memory [48] because this identification often requires elevated privilege and is prone to false positives. We assume that the installed malware can be removed from the system by security-conscious users in a timely manner, so that even though sensitive login information stored by BPMs can be stolen within a short period of time, it is very difficult for attackers to use tools such as keyloggers to further intercept users' passwords for a long period of time. One typical example is that anti-malware programs such as Microsoft Forefront Endpoint Protection may detect the infection, report the suspicious file transmission, and finally remove the malware and infected files. Another typical example is that solutions such as the Back to the Future framework [49] can restore the system to a prior good state and preserve system integrity. The users can then have the opportunities to install security patches and enforce stricter security policies on their systems. A similar assumption is also made in other systems such as Google's 2-step verification system [50].

We also assume that domain name systems are secure and reliable and we do not specifically consider pharming attacks. This assumption is made in all the BPMs and we believe pharming and other DNS attacks should be addressed by more general solutions. Similarly, we do not consider other Web attacks such as cross-site scripting that can also steal sensitive login information because those attacks have their own specific threat models and assumptions.

3.2. The Essential Problem and An Analogy

The essential problem is that the encrypted passwords stored by BPMs of the five most popular browsers are very weakly protected in many situations. In our investigation, we found without the protection of a master password mechanism, the encrypted passwords stored by the five BPMs (Table 1) can be trivially decrypted by attackers for logging into victims' accounts on the corresponding websites. We have developed tools and verified this severe vulnerability of the latest versions (by March 2014 as shown in Table 1) of the five BPMs on Windows 7. This vulnerability is common to all these browsers because the keys used by these browsers for encrypting/decrypting a user's login information can be easily extracted or generated by attackers. The decrypted login information can be easily sent out to attackers and the entire attack could be finished in one second. In the cases when a master password is used by a user in Firefox (Table 1), the problem is that even though decrypting a user's login information becomes harder, brute force attacks and phishing attacks against the master password are still quite possible. We believe that it is critical for users to choose strong master passwords, and it is also critical for BPMs to properly use and protect master passwords.

We term these problems as ***vulnerabilities*** because they are security design weaknesses of existing BPMs that can be exploited by popular attacks such as drive-by downloads [42, 43, 44, 45, 46]; we do not mean these existing BPMs do not work as they were designed.

A BPM is analogous to a safe, and a master password is analogous to the combination to the safe. The current reality is that the "safe" of Google Chrome, Internet Explorer, and Safari does not allow a user to set a "combination" at all. Our decryption tools can easily and accurately open the "safe". Firefox allows a user to set a "combination", but does not make it mandatory. Our decryption tools can also easily and accurately open the "safe" of Firefox if a "combination" was not set. For example, using drive-by downloads, an attacker can deliver our decryption tools to a user's computer and trigger their execution. In one second, all the passwords and usernames saved by BPMs can be completely decrypted and sent back to the attacker's website or email account. The malware detector installed on the user's computer may report suspicious activities, and the user may immediately take actions to disable the Internet connection. But it could be too late! With a successful drive-by download, attackers can perform many types of malicious activities. However, similar to burglars, if attackers know they can easily open the "safe", they would like to first steal the most valuable items from the "safe" within a short period of time.

3.3. Without a Master Password Mechanism

Through source code analysis, binary file analysis, and experiments, we found that Firefox uses the three-key Triple-DES algorithm to encrypt a user's passwords for different websites. ***Firefox*** saves each encrypted username, encrypted password, and plaintext login webpage URL address into the *login* table of a SQLite [51] database file named *signons.sqlite*. The Triple-DES keys are generated once by Firefox and then saved into a binary file named *key3.db* starting from the byte offset location 0x2F90. Although the keys generated on different computers are different, they are not bound to a particular computer or protected by other mechanisms. Therefore, as verified by our tools, an attacker can simply steal both the *signons.sqlite* file and the *key3.db* file and then accurately decrypt every username and password pair on any computer.

In their latest Window 7 versions, all the other four browsers Internet Explorer, Google Chrome, Safari, and Opera use the Windows API functions *CryptProtectData* [52] and *CryptUnprotectData* [53] to perform encryption and decryption, respectively. The key benefit of using these two functions is that "typically, only a user with the same logon credential as the user who encrypted the data can decrypt the data [52]." To use these two API functions, an application (e.g., a browser) does not generate or provide encryption/decryption keys because the symmetric keys will be deterministically generated in these two functions based (by default) on the profile of the current Windows user. An application can use the *dwFlags* input parameter to specify that the keys should be simply associated with the current computer; it can also use the *pOptionalEntropy* input parameter to provide additional entropy to the two functions.

We found ***Google Chrome*** saves each plaintext username, encrypted password, and plaintext login webpage URL address into the *logins* table of a SQLite [51] database file named *Login Data*. Google Chrome does not provide additional entropy to the two API functions. ***Opera*** (version 20.0) uses the identical mechanism as that of Google Chrome, although its early versions such as version 12.02 used a different mechanism [40]. ***Safari*** saves each plaintext username, encrypted password, and plaintext login webpage URL address into a special property list file named *keychain.plist*. Safari provides a static 144-byte salt as the additional entropy to the two API functions. ***Internet Explorer*** encrypts each username and password pair and saves the ciphertext as a *value data* under the Windows registry entry: "HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\IntelliForms\Storage2\". Each saved *value data* can be indexed by a *value name*, which is calculated by hashing the login webpage URL address. Internet Explorer also provides the login webpage URL address as the additional entropy to the two API functions.

We found all these four browsers set the *dwFlags* input parameter value as the default value zero, which means that the symmetric keys are associated with each individual Windows 7 user. Therefore, it is not very easy for attackers to decrypt the stolen ciphertexts on another computer or using another Windows account. However, attackers who can steal the ciphertexts (for example, using bots [13] or Trojan horses) can simply decrypt the ciphertexts on the victim's machine when the victim is logged into the Windows; then, the decrypted login information can be directly sent back to attackers. We have developed tools that can decrypt the ciphertexts stored by all these four browsers. In more details, for Google Chrome, our tool selects each record from the *logins* table of the *Login Data* SQLite database, converts the encrypted password from the SQLite BLOB [51] type to a string type, and supplies the encrypted password to the *CryptUnprotectData* [53] function. The decryption tool for Opera version 20.0 is identical to that for Google Chrome, and we also have the decryption tool for Opera version 12.02 [40]. For Safari, our tool converts the *keychain.plist* property list file to an XML document, parses the XML document to obtain each encrypted password, and supplies the encrypted password and that static 144-byte salt to the *CryptUnprotectData* function. For Internet Explorer, our tool hashes the popular login webpage URL addresses contained in a dictionary, queries the Windows registry using each hashed URL address to identify a matched *value name*, and supplies the associated *value data* and the corresponding login webpage URL address (as the additional entropy) to the *CryptUnprotectData* function.

3.4. With a Master Password Mechanism

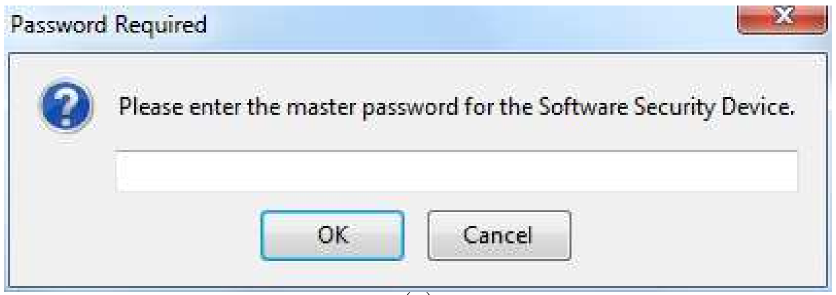
The BPM of Firefox allows a user to set a master password (Table 1) to further protect the encryption keys or encrypted passwords. In ***Firefox***, the master password and a global 160-bit salt will be hashed using a SHA-1 algorithm to generate a master key. This master key is used to encrypt those three Triple-DES keys before saving them to the *key3.db* file. Firefox also uses this master key to encrypt a hard-coded string "password-check" and saves the ciphertext to the *key3.db* file; later, Firefox will decrypt this ciphertext to authenticate a user before further decrypting the three Triple-DES keys.

Using a master password can better protect the stored passwords in Firefox. However, a master password mechanism should be carefully designed to maximize security. One main security concern is the brute force attacks against the master password. For one example, if the computation time for verifying a master password is very small as in Firefox (which rejects an invalid master password in one millisecond), it is still possible to effectively perform brute force attacks against a user's master password. For another example, encrypting the hard-coded "password-check" string in Firefox for user authentication does not increase security and may actually decrease security in the case when both the *signons.sqlite* file and the *key3.db* file (containing the 160-bit salt) are stolen. Although decrypting the Triple-DES keys is still very difficult if the master password is unknown, an attacker can simply bypass this user authentication step using an instrumented Firefox. Moreover, this hard-coded plaintext and its ciphertext encrypted by the master key can also be used by an attacker to verify the correctness of dictionary or brute-force attacks against the master password.

Another main security concern is the phishing attacks against the master password. Figure 1(a) illustrates the genuine master password entry dialog box in Firefox, which will be displayed to a user for the first autofill operation in a browsing session. Figure 1(b) illustrates one fake master password entry dialog box created by the JavaScript *prompt()* function. Such a fake dialog box can be displayed by any regular webpage on all the five browsers without being blocked by browsers' "block pop-up windows" options because it is not a separate HTML document window. We speculate that even such a simple spoofing technique can effectively obtain master passwords from vulnerable users. Indeed, a regular webpage can also use JavaScript and CSS (Cascading Style Sheets) to create sophisticated dialog boxes that are more similar to a genuine master password entry dialog box. Such attacks are similar to the Web-based spoofing attacks on OS password-entry dialogs illustrated by Bravo-Lillo et al. [54] and the Web single sign-on phishing attacks illustrated by Yue [38]. Overall, our position is that a BPM should not use these types of easy-to-spoof master password entry dialog boxes at all, and should not frequently ask a user to enter the master password in a single browsing session.

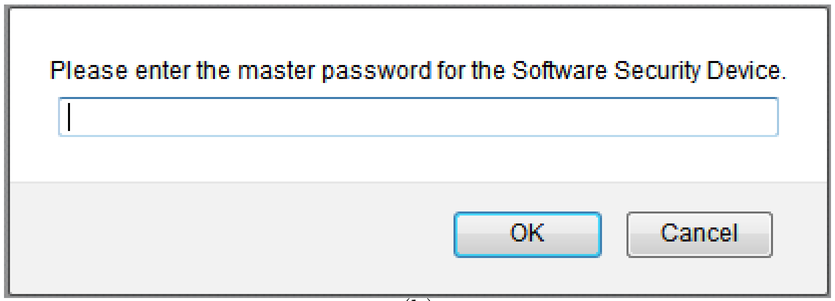
4. CSF-BPM Design

We now present the design of the Cloud-based Storage-Free BPM (CSF-BPM). It is cloud-based storage-free in the sense that the protected data will be completely stored in the cloud – nothing needs to be stored on a user's computer. We want to move the storage into the cloud for two key reasons. One is that in the long run trustworthy storage services in the cloud [14, 15, 16, 17, 18, 19] can better protect a regular user's data than local computers (which may not be timely and properly patched) do, especially if a storage service uses secret sharing schemes such as the (k, n) threshold scheme [20] to only save pieces of the encrypted data to different cloud vendors [14]. The other reason is that the stored data can be easily accessible to the user across


Password Required

Please enter the master password for the Software Security Device.

(a)


Please enter the master password for the Software Security Device.

(b)

Figure 1: The (a) genuine and (b) fake master password entry dialog box in Firefox.

different OS accounts on the same computer and across computers at different locations at anytime. This design differs from the BPM designs of all the five most popular browsers. Based on the threat model and assumptions defined in the last section, we design CSF-BPM to synthesize the desired security properties such as confidentiality, integrity, and availability.

4.1. High-level Architecture

Figure 2 illustrates the high-level architecture of CSF-BPM. The BPM of the browser simply consists of a User Interface (UI) component, a Record Management (RM) component, a Record Generation (RG) component, a Record Decryption (RD) component, and a record synchronization (Sync) component. The UI component will provide configuration and management interfaces accessible at a single location. The BPM itself does not include any persistent storage component such as a file or database; instead, it will generate Encrypted Login Information Records (***ELIRs***), save *protected ELIRs* to a Secure and Reliable Storage (SRS) service in the cloud, and retrieve protected ELIRs in real-time whenever needed. Such a generic BPM design can be seamlessly integrated into different browsers.

An SRS service simply needs to support user authentication (e.g., over HTTPS) and per-user storage so that its deployment in the cloud can be easily achieved. For example, the synchronization service associated with Firefox or Google Chrome (Table 1) could be directly used as an SRS service without making any modification. The SRS service will store a Per-User Protected ELIRs (***PUPE***) data object (to be illustrated in Figure 5) for each SRS user. The communication protocol between the BPM and SRS is also very simple: after a user is authenticated to SRS, the Sync component of BPM will transparently send HTTPS requests to SRS to *retrieve* or *save* the protected ELIRs of the user. An SRS service should be highly reliable and available. However, to further increase reliability and availability, the BPM can store protected ELIRs to multiple independent SRS services. One of them is used as the primary SRS service; others will be used as secondary SRS services. The Sync component of BPM will transparently synchronize protected ELIRs from the primary SRS service to secondary SRS services.

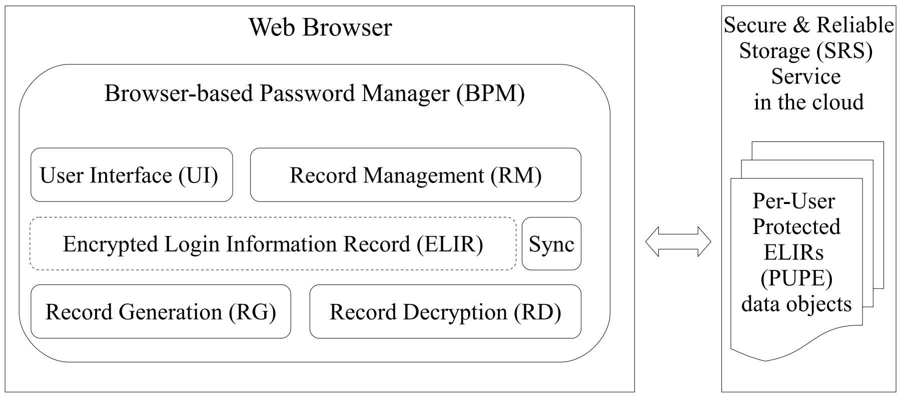

Web Browser
Browser-based Password Manager (BPM)
User Interface (UI)
Record Management (RM)
Encrypted Login Information Record (ELIR)
Sync
Record Generation (RG)
Record Decryption (RD)
Secure & Reliable Storage (SRS) Service in the cloud
Per-User Protected ELIRs (PUPE) data objects

Figure 2: High-level architecture of the Cloud-based Storage-Free BPM (CSF-BPM).

4.2. Design Details

In the following subsections, we detail the basic usage of CSF-BPM, the ELIR record, the key derivation and password encryption process, the PUPE data object, and the password decryption process.

4.2.1. Basic Usage

To use CSF-BPM, a user needs to remember a Single Strong Master Password (***SSMP***) with the strength [55, 56] assured by the traditional proactive password checking techniques and certain length requirements [57, 58, 59], or by the latest reactive proscriptive intervention techniques [60]. Using a master password is also advocated in other proposed systems such as Nigori [61]. The user also needs to set up an account (srsUsername, srsPassword) on an SRS service and configure this service once through the UI component of BPM.

At the beginning of each browsing session, the user needs to authenticate to the SRS service and provide the SSMP to BPM. After that, BPM will take care of everything else such as triggering the remembering of website passwords, encrypting and decrypting ELIRs, and triggering the autofill of passwords. Both the srsUsername and srsPassword pair and SSMP need be provided only once in a session through the special UI component of BPM. This requirement adds some burden to users in exchange of the increased security. This special UI component is integrated into the configuration UI of Firefox, thus cannot be easily spoofed by JavaScript (e.g., using the *prompt()* function) on regular webpages. Meanwhile, CSF-BPM can detect and require that the SSMP to be different from the srsPassword and any website password. These design choices could be helpful in protecting SSMP against phishing attacks. Note that if multiple SRS services are used, providing the srsUsername and srsPassword for each SRS service at the beginning of each session may be unwieldy; we will investigate the potential of using password hashing techniques [31, 32, 33] to address this issue in the future.

4.2.2. ELIR Record

The basic format of an ELIR record is shown in Figure 3. Here, recordSalt is a large and sufficiently random per-record salt generated by BPM. It is used to calculate the symmetric record key (denoted recordKey) for encrypting a user's plaintext password (denoted sitePassword) for an account (denoted siteUsername) on a website (with siteURL as the login webpage URL address). The recordKey can be deterministically generated by using a password-based key derivation function such as PBKDF2 specified in the PKCS5 specification version 2.0 [62]. The basic format of an ELIR record can also include the IDs (or names) of the username and password fields in the login webpage, and it can be further extended if necessary.

4.2.3. Key Derivation and Password Encryption

Using PBKDF2 [62], our SSMP-based key derivation and password encryption process consists of five steps illustrated in Formulas 1, 2, 3, 4, and 5

siteURL	siteUsername	encryptedSitePassword
recordSalt	…..	

Figure 3: The basic format of an Encrypted Login Information Record (ELIR).

in Figure 4. The input parameters mainSalt and aeSalt in Formulas 1 and 2 are large and sufficiently random per-user salts generated by BPM at the first time when a user authenticates to the SRS service through the UI component of BPM. In Formulas 1, 2, and 3, the input parameters c1, c2, and c3 represent iteration counts for key stretching; the input parameters dkLen1, dkLen2, and dkLen3 represent lengths of the derived keys, and they are related to the underlying pseudorandom function used in the PBKDF2 implementation.

$$mainKey = PBKDF2(SSMP, mainSalt, c1, dkLen1) \tag{1}$$

$$aeKey = PBKDF2(mainKey, aeSalt, c2, dkLen2) \tag{2}$$

$$recordKey = PBKDF2(mainKey, recordSalt, c3, dkLen3) \tag{3}$$

$$encryptedSitePassword = E(recordKey, sitePassword) \tag{4}$$

$$protectedELIRs = AE(aeKey, concatenatedELIRs) \tag{5}$$

Figure 4: Formulas in the SSMP-based key derivation and password encryption process.

The salts and iteration counts in PBKDF2 are used to secure against dictionary and brute-force attacks, and they need not be kept secret [62]. The strength of SSMP also helps secure against these two types of attacks. In Formula 1, a mainKey is calculated and will be used in each browsing session. SSMP is typed only once and will be erased from memory after mainKey is calculated. In Formula 3, a unique recordKey is generated (using the per-record recordSalt) for each website account of the user. In Formula 4, a NIST-approved symmetric encryption algorithm E such as AES [63] (together with a block cipher mode of operation if the sitePassword is long) can be used to encrypt the sitePassword. In Formula 5, a NIST-approved Authenticated Encryption block cipher mode AE such as CCM (Counter with CBC-MAC) [64] can be used to simultaneously protect confidentiality and authenticity (integrity) of the *concatenatedELIRs* (i.e., the concatenated string of all the ELIR records) of an SRS user. The aeKey used here is generated by Formula 2.

The iteration count c1 used in Formula 1 should be large so that the mainKey calculation will take a few seconds; therefore, brute force attacks against SSMP become computationally infeasible. But c1 should not be too large to make a user wait for a long period of time at the beginning of a session. Iteration counts c2 and c3 should not be too large so that generating aeKey and recordKey would not cause a user to perceive any delay.

The mainKey is kept in memory in the whole browsing session. Identifying a single key in the memory is more difficult than identifying a block of key materials with structural information [48]. Therefore, the aeKey and recordKey are scrubbed immediately after use so that less structural information (i.e., the keys and the related website information) will be left in the memory for attackers to exploit. Although Formula 5 will simultaneously protect confidentiality and authenticity (integrity) of the concatenatedELIRs of an SRS user, encrypting each sitePassword in Formula 4 is still important. This is because the concatenatedELIRs is also kept in memory in the whole browsing session. In comparison with the mainKey which is basically a random-looking value, the structure of ELIR records and concatenatedELIRs can be easily identified from memory. Therefore, assuming an attacker cannot easily identify the mainKey but can easily identify ELIR records (which contain structural information) from memory, it is still computationally infeasible for the attacker to crack each individual recordKey and decrypt the corresponding sitePassword.

The iteration counts c1, c2, and c3 can be adjusted by BPM with or without user intervention to maximize security while minimizing inconvenience to users [65]. In our current design, CSF-BPM adaptively computes the maximum values of iteration counts based on the specified computation times for Formulas 1, 2, and 3, respectively. For example, if a 10-second computation time is specified for deriving the mainKey, CSF-BPM will run Formula 1 for 10 seconds to derive the mainKey and meanwhile finalize the corresponding c1 value. Such a scheme allows CSF-BPM to easily maximize the security

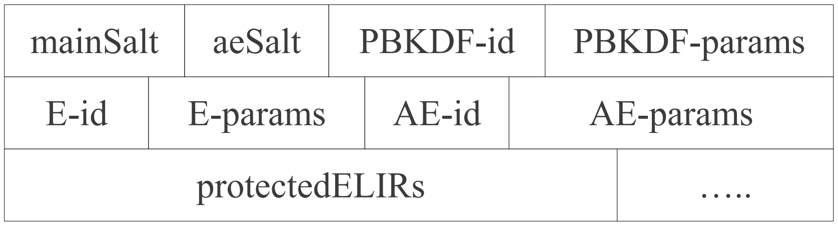


Figure 5: The Per-User Protected ELIRs (PUPE) data object saved for each SRS user.

strength of key derivation within a specified delay limit on each individual computer.

Overall, all the computations including salt generation, key derivation, and encryption etc. are performed on BPM. Neither the SSMP nor any derived cryptographic key will be revealed to an SRS service or a third party. An SRS service does not need to provide any special computational support to BPM; it simply needs to save a PUPE data object for each SRS user.

4.2.4. PUPE Data Object

As illustrated in Figure 5, each PUPE object contains the protectedELIRs (Formula 5) of an SRS user and all the algorithm related information. Here, PBKDF-id specifies the identifier for the PBKDF2 key derivation function [62]; PBKDF-params specify the PBKDF2 parameters such as c1, c2, c3, dkLen1, dkLen2, and dkLen3 used in Formulas 1, 2, and 3. E-id and E-params specify the identifier and parameters, respectively, for the symmetric encryption algorithm (and the mode of operation) used in Formula 4. AE-id and AE-params specify the identifier and parameters, respectively, for the authenticated encryption block cipher mode used in Formula 5. For example, if AE-id specifies the CCM authenticated encryption block cipher mode [64], then AE-params will contain the Nonce and the Associated Data input parameters used by CCM. Each PUPE data object can be simply saved as a binary or encoded string object for an SRS user because its structure does not need to be known or taken care of by any SRS service. Such a PUPE data object design makes the selection of algorithms and the selection of SRS services very flexible.

4.2.5. Password Decryption

To decrypt the saved website passwords for autofill, BPM will perform five steps: (1) retrieve the PUPE data object saved for the SRS user; (2)

generate the mainKey and aeKey using Formulas 1 and 2; (3) decrypt and verify the protectedELIRs using the reverse process of Formula 5 such as the CCM Decryption-Verification process [64]; (4) obtain the recordSalt of each ELIR and generate the recordKey using Formula 3; (5) finally, decrypt the encryptedSitePassword using the reverse process of Formula 4. Note that at step (3), both the integrity of the protectedELIRs and the authenticity of the BPM user are verified because the success of this step relies on using the correct SSMP. Also at this step, siteURL and siteUsername of all the ELIRs can be obtained by BPM to determine whether this user has previously saved login information for the currently visited website. Normally, the first three steps will be performed once for the entire browsing session, and the last two steps will be performed once for each website that is either currently visited by the user, or its domain name is queried by the user to simply look up the corresponding username and password. In comparison with the password manager of Firefox, CSF-BPM uses the steps (2) and (3) to ensure a much stronger confidentiality and integrity guarantee, even if attackers can steal the retrieved PUPE object.

Because all the salts are randomly generated by BPM, the protectedELIRs saved to different SRS accounts or different SRS services will be different. BPM can transparently change mainSalt, aeSalt, and every recordSalt whenever necessary. A user also has the flexibility to change SSMP and any sitePassword whenever necessary. In these cases, all what need to be done by BPM is to update the new PUPE data object and ELIRs to each corresponding SRS service account. A user can also flexibly change any srsPassword, which is completely independent of SSMP.

4.3. Design Rationales and Security Analysis

We now further justify the important design rationales of CSF-BPM by focusing on analyzing its confidentiality, integrity, and availability security properties, and by comparing its design with other design alternatives especially the BPM design of Firefox that also provides a master password mechanism.

In terms of the confidentiality, first, by having a unique cloud-based storage-free architecture, CSF-BPM can in the long run effectively reduce the opportunities for attackers to steal and further crack regular users' saved website passwords. Second, even if attackers (including insiders of an SRS service) can steal the saved data, it is computationally infeasible for attackers to decrypt the stolen data to obtain users' login information for different websites. CSF-BPM provides this security guarantee by mandating a strong SSMP that satisfies certain strength requirements [57, 58, 59], by using the PBKDF2 key derivation function [62] with randomly generated salts and adaptively computed large iteration counts, and by following NIST-approved symmetric encryption [63] and authenticated encryption [64] algorithms. Basically, even if attackers can steal the saved data, they have to guess (albeit stealing attacks are still possible as discussed in Section 7) a user's strong SSMP in a very large space determined mainly by the length and character set requirements of SSMP with each try taking seconds of computation time.

We can estimate the effort of brute force attacks based on the computational power exemplified in a very popular cryptography textbook [66] authored by William Stallings. The Table 3.5 (chapter 3, page 78, and 6th edition) of this textbook shows that a rate of 1 billion (10^9) decryptions per second can be achieved by today's multicore computers. For simplicity but without loss of generality, we consider that SHA-1/SHA-2 [67] hash operations can be performed at the similar rate by multicore computers, although this is a conservative consideration because a hash operation is normally more efficient than a decryption operation. 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. The search space for an 8-character master password will be 62^8. Therefore, it will take on average 30.3 hours for a multicore computer to successfully perform a brute force attack against a user's 8-character master password used in Firefox. However, with the c1 value as 300,000, CSF-BPM increases the brute force effort to 300,000 times of 30.3 hours, that is about 1038 years for the same multicore computer. C++ implementation of PBKDF2 would be more efficient, thus allowing the c1 value to be increased while still maintaining the 10-second limitation. Increasing c1 would make offline brute-force attacks harder.

In terms of the integrity, the NIST-approved CCM authenticated encryption algorithm [64] enables CSF-BPM to accurately detect both any invalid SSMP try and any modification to a saved PUPE data object. Moreover, this detection is securely performed in the sense that attackers cannot take advantage of it to effectively conduct brute force attacks against the SSMP.

In terms of the availability, an SRS simply needs to be a storage service in the cloud and it does not need to provide any special computational support. Such a design decision makes it very easy to either use an existing storage service in the cloud as an SRS service or deploy a new SRS service by an organization. CSF-BPM supports multiple SRS services and it uses a simple

HTTPS-based communication protocol; these design decisions also further enhance the availability.

CSF-BPM offers a better security in comparison with the BPM of Firefox that also provides a master password mechanism. Firefox saves the encrypted data locally on a user's computer and does not use strong key derivation functions (Section 3.4); thus, its confidentiality assurance is weak in consideration of brute force attacks. Firefox can detect an invalid master password try, but the detection mechanism is not secure (Section 3.4). Firefox does not detect any modification to the saved data; the modified data will still be decrypted into incorrect and often non-displayable values, but no information is provided to a user. In addition, the synchronization mechanism of Firefox is tightly bound to Mozilla's own server [68]; thus, the availability of the saved data is not well assured by the BPM.

It is also worth mentioning that CSF-BPM does remain as vulnerable as existing BPMs to the attacks ruled out by our assumptions defined in the threat model (Section 3.1). For example, malware that persists on a user's computer could obtain the user's passwords from any BPM, and a malicious domain name query result could let any BPM auto-fill a user's passwords to attackers' websites.

Other cloud-based password system design alternatives also exist, but they often have different design objectives and limitations. For one example, Passpet [33] can help a user generate passwords for different websites based on a master password. Similar to Password Multiplier [31], Passpet is essentially a password generator instead of a password manager because it uses password hashing techniques to deterministically generate website passwords instead of remembering users' original passwords. Requiring users to migrate their original passwords to hashed passwords is a biggest limitation of hashing-based password generation solutions as acknowledged in the Password Multiplier paper [31]. In addition, Passpet imposes very special requirements on its remote storage server: the SRP authentication protocol [69] must be used and some specific commands must be supported. These requirements limit the deployability of Passpet. For another example, LastPass [34] and RoboForm [23] are two very popular cloud-based BPMs. However, both of them have severe security flaws such as very vulnerable to insider attacks, local decryption attacks, and brute-force attacks; we refer readers to our recent paper [35] for the details.

5. Implementation

CSF-BPM is designed to be implementable in different Web browsers and to be able to easily use different SRS services. In this section, we briefly describe some important details of our Firefox version CSF-BPM implementation; we hope these details can be helpful for others to integrate CSF-BPM into more browsers.

We have implemented a CSF-BPM system and seamlessly integrated it into the Firefox Web browser. This system can directly use the Firefox Sync server operated by Mozilla [68] as an SRS service without making any modification to this server; thus, a free of charge SRS service is directly available to users. It is important to note that in our implementation, the interfaces for triggering the remembering and autofill of passwords in Firefox (Section 2.2) are not changed; only the operations happening behind the scenes are changed.

Figure 6 illustrates more details about our Firefox implementation of CSF-BPM. We mainly implemented two new modules in Firefox: an In-memory Record Management Service (ImRMS) and a Key Management Service (KMS). Both modules are implemented as JavaScript version XP-COM (Cross-Platform Component Object Model) [70] components and run as services in Firefox. ImRMS is responsible for generating the PUPE object, uploading or retrieving the PUPE object, and maintaining all the ELIR records. In essence, ImRMS replaced the original persistent password storage of Firefox (Section 3) with an in-memory ELIR array and its corresponding add/delete/modify/search interface. KMS is responsible for generating salts, deriving keys, and preparing for other parameters used in Formulas 1, 2, 3, 4, and 5. In our current implementation, all the salts are 128-bit random numbers, and the default length of all those keys is also 128-bit. However, we can easily change the default length to 256-bit for all the salts and keys. Currently, we used the PBKDF2 [62], CCM [64], and AES [63] implementations provided in the Stanford JavaScript Crypto Library [71].

In addition to implementing ImRMS and KMS, we also made some important modifications to the Sync module in Firefox. In the original Sync module, a 26-character ***recovery key*** will be generated when a user creates a Sync account. This recovery key is not shared with the Firefox Sync server, and it is mainly used to protect other cryptographic keys that are stored on the Firefox Sync server for a user. A user must save this recovery key and provide it to Firefox on different computers together with the

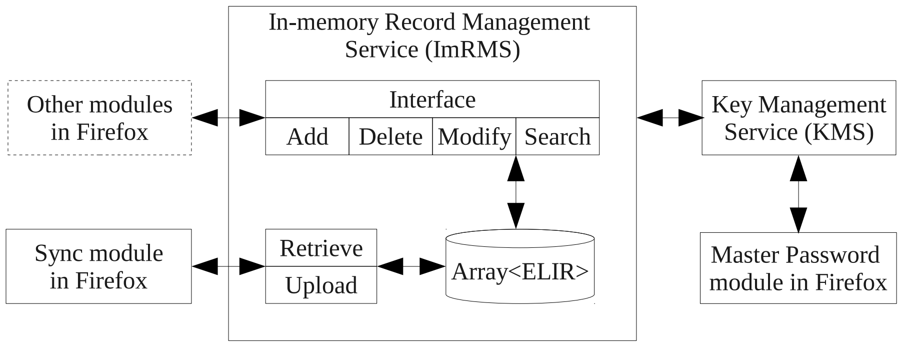

In-memory Record Management Service (ImRMS)
Interface
Add
Delete
Modify
Search
Other modules in Firefox
Key Management Service (KMS)
Sync module in Firefox
Retrieve
Upload
Array<ELIR>
Master Password module in Firefox

Figure 6: Detailed implementation of CSF-BPM in Firefox.

Sync account username and password whenever the Sync mechanism needs to be used. This requirement limits the usability of the Sync mechanism; meanwhile, the recovery key is not needed at all in the CSF-BPM design. Therefore, in our implementation, one main modification to the Sync module is removing the dependence of using Firefox Sync server on recovery key for the password manager feature. As a result, a user does not need to save and provide the recovery key at all if he or she uses CSF-BPM and uses the Firefox Sync server as the SRS service. The other main modification is that we use a Weave Basic Object (WBO) [68] assigned to the default Mozilla *passwords collection* to store the PUPE object in the Firefox Sync server. Both modifications are specific to using the Firefox Sync server as the SRS service.

6. Evaluation

We built the Firefox version CSF-BPM on a Ubuntu Linux system. We tested the correctness of our implementation and its integration with the Firefox Web browser, we intensively evaluated its performance, and we also evaluated its usability through a user study.

6.1. Correctness

We selected 30 websites as listed in Table 2 to perform the correctness verification. Most of the websites were selected from the top 50 websites listed by Alexa.com; however, we removed non-English websites, gray content websites, and the websites that did not allow us to create an account. We also selected some of our frequently used websites.

Table 2: The 30 websites.

mail.google.com	facebook.com	mail.yahoo.com
wikipedia.com	twitter.com	amazon.com
linkedin.com	wordpress.com	ebay.com
fc2.com	craigslist.org	imdb.org
aol.com	digg.com	careerbuilder.com
buy.com	aaa.com	newegg.com
tumblr.com	alibaba.com	4shared.com
cnn.com	nytimes.com	foxnews.com
weather.com	groupon.com	photobucket.com
myspace.com	webmail.uccs.edu	portal.prod.uccs.edu

On each website, we went through four main steps. First, we opened Firefox and typed an SRS account (i.e., a Firefox Sync account) and SSMP. Second, we logged into the website and confirmed to save the website password. Third, we logged out the website and logged into it again with the auto-filled password. Finally, we closed Firefox, re-opened Firefox, typed the SRS account and SSMP, and logged into the website again with the auto-filled password.

Through the execution of those steps, we verified that our implementation works precisely as designed; meanwhile, it integrates smoothly with Firefox and does not cause any logic or runtime error. In more details, we observed that CSF-BPM can correctly save and auto-fill passwords on all those websites. It also works correctly in the situation when two or more accounts on a website are used. In addition, it does not affect the functionality of other features in Firefox such as the form autocomplete feature and the Sync feature. We also verified that nothing is saved to the original persistent password storage of Firefox.

We have two other observations in our experiments. One is that some other websites share the same siteURL (i.e., the login webpage URL) values with the websites listed in Table 2. For example, youtube.com and mail.google.com share the same siteURL, flickr.com and mail.yahoo.com share the same siteURL, and msn.com and live.com share the same siteURL. The evaluation results are correct on those websites for both CSF-BPM and the original Firefox BPM. The other observation is that some other websites such as paypal.com and wellsfargo.com set the *autocomplete="off"* property on their password fields or login forms; therefore, passwords will not be saved at all by BPMs including our CSF-BPM.

6.2. Performance

We performed both micro-benchmark experiments and macro-benchmark experiments to evaluate the performance of CSF-BPM. In these experiments, we ran CSF-BPM on a desktop computer with 2.33GHz CPU, 3.2 GB memory, and 100Mbps network card. All the experiments were repeated 5 times and we present the average results.

6.2.1. Micro-benchmark Experiments

In micro-benchmark experiments, we ran CSF-BPM using scripts to evaluate the following four aspects of performance.

(a) Key derivation: We mentioned in Section 4 that CSF-BPM adaptively computes the maximum values of the iteration counts c1, c2, and c3 based on the specified computation times for Formulas 1, 2, and 3, respectively. Those three formulas have the same performance because in our implementation they use the same PBKDF2 [62] algorithm, same salt length, and same key length. The performance impact of different SSMP lengths in Formula 1 is negligible because the intermediate values will have the same length as the key length after the first iteration. Therefore, in our experiments, we simply increased the computation time of the PBKDF2 algorithm from one second to 20 seconds to calculate the iteration count values.

Overall, the iteration count values increase linearly with the increasing of the computation time. The larger the iteration counts, the more secure the derived keys [62]. As suggested in RFC 2898 [62] in year 2000, "A modest number of iterations, say 1000, is not likely to be a burden for legitimate parties when computing a key, but will be a significant burden for opponents." This suggested number should definitely be increased with the increasing computing powers of potential attackers [65, 72]. Currently, CSF-BPM uses 10 seconds, one second, and one second as the default times for adaptively computing iteration counts c1, c2, and c3, respectively. Correspondingly, the value of c1 is around 300,000 and the values of c2 and c3 are around 30,000 on our test computer. We chose 10 seconds as the default computation time of c1 to impose a significant SSMP guessing burden on attackers. Asking a user to wait for 10 seconds once at the beginning of a browsing session is still acceptable as shown in our user study in Section 6.3, but this waiting time should not be too long taking the usability in consideration [1, 73].

(b) Password encryption and decryption: This performance refers to Formula 4 and its reverse process. In our experiments, we observed that the JavaScript implementation of AES [63] provided in the Stanford JavaScript

Crypto Library [71] can consistently encrypt and decrypt one 16-byte block within one millisecond (ms).
(c) concatenatedELIRs encryption and decryption: This performance refers to Formula 5 and its reverse process, more specifically, the CCM Authentication-Encryption process and its reverse Decryption-Verification process [64]. In our experiments, we varied the total number of randomly generated ELIR records (Figure 3) from one to 400. We observed that both the size of concatenatedELIRs and the size of PUPE increase linearly with the increasing of the number of ELIRs. The size of PUPE for 400 records is 107KB, which is much smaller than the size (448KB) of the physical SQLite database file (*signons.sqlite*) in Firefox for 400 records. We observed that the CCM Authentication-Encryption process and the CCM Decryption-Verification process can be performed within 25.8 milliseconds and 30.6 milliseconds, respectively, for the concatenatedELIRs that contains 400 records.
(d) PUPE upload and retrieval: We evaluated the PUPE upload and retrieval time on the Firefox Sync server. The upload time is the round-trip time between CSF-BPM sends a POST type of HTTPS request to the Firefox Sync server to upload a PUPE object and it receives the corresponding response of that HTTPS request. The retrieval time is the round-trip time between CSF-BPM sends a GET type of HTTPS request to the Firefox Sync server to retrieve a PUPE object and it receives the corresponding PUPE object. As shown in Figure 7, with the increasing number of ELIR records in the PUPE object, the PUPE upload time increases steadily with the peak value at about 900 milliseconds for 400 records, and the PUPE retrieval time increases steadily with the peak value at about 700 milliseconds for 400 records. Note that 400 is a very large number because on average a user has less than 30 online accounts as measured in [12]. Therefore, overall, both the PUPE upload time and retrieval time are short and acceptable.

6.2.2. Macro-benchmark Experiments

We further measured the PUPE upload and retrieval time in real usage scenarios. We still use the 30 websites listed in Table 2 to perform the experiments. We visited those 30 websites one by one to let CSF-BPM incrementally record website passwords. The results show that the PUPE upload time is less than 330 milliseconds in all the cases, and the PUPE retrieval time also stays around 200 milliseconds. In each Web browsing session, the PUPE retrieval operation is performed only once, and the PUPE upload operation is performed only when the PUPE object is created or updated.

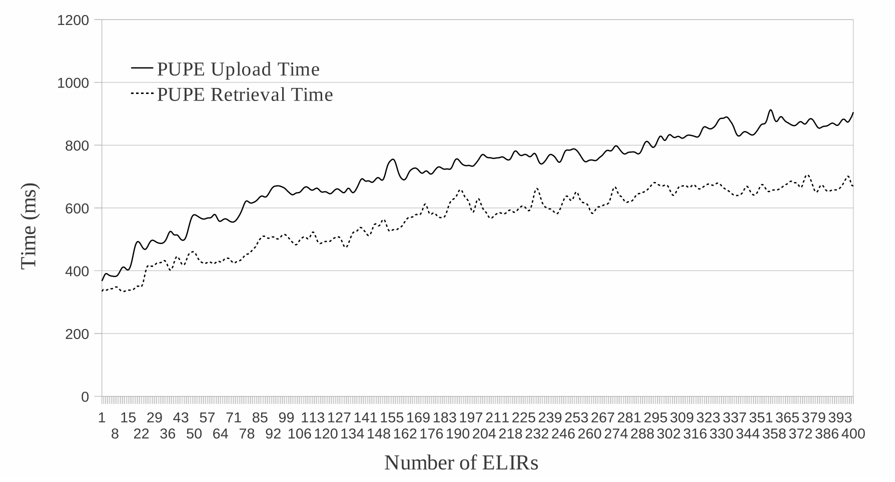


Figure 7: PUPE upload and retrieval time vs. the number of ELIR records in the PUPE object.

Therefore, these performance results in the realistic usage scenarios further demonstrate that CSF-BPM can efficiently use the Firefox Sync server as an SRS service. Indeed, we did not observe any noticeable delay in this set of macro-benchmark experiments.

6.3. Usability

To evaluate the usability of the Firefox version CSF-BPM, we conducted a user study. To be fair, we compared the usability between our Firefox version CSF-BPM and the original password manager of Firefox that uses both the master password and the Sync mechanism. We mainly measured whether there are statistically significant usability differences between using our Firefox version CSF-BPM and using the original password manager of Firefox. This user study was pre-approved by the IRB (Institutional Review Board) of our university.

6.3.1. Participants

Thirty adults, 15 females and 15 males, participated in our user study. They were voluntary students (9), faculty members (1), staff members(3), and general public members (17) randomly recruited on our campus library, bookstore, and cafeteria, etc.; they came from 14 different majors. Eighteen participants were between ages of 18 and 30, and twelve participants were over 30 years old; we did not further ask their detailed ages. All the participants claimed that they use computers and Web browsers daily, and five of them claimed that they use the password manager of Firefox, Google Chrome, or Opera to manage their online passwords. We did not collect any other demographic or sensitive information from participants. We did not screen participants based on any of their Web browsing experience. We did not provide monetary compensation to the participants.

6.3.2. Scenario and Procedure

On a Ubuntu Linux system, we installed an original Firefox as Firefox-A, and installed another Firefox with our CSF-BPM as Firefox-B. To have a fair comparison, we only told participants that there are two different password managers in two Firefox browsers (Firefox-A and Firefox-B), but we did not tell them which one is the original Firefox and which one is ours.

We asked each participant to perform two procedures: Procedure-A and Procedure-B. In Procedure-A, a participant uses Firefox-A to first perform an *Initial Visit* scenario on one computer to let the password manager of Firefox remember the accounts of three testing websites (mail.yahoo.com, www.amazon.com, and www.facebook.com), and then perform a *Revisit* scenario on another computer (i.e., using CSF-BPM on a new computer) to let the password manager automatically fill the login forms on the three visited testing websites. In Procedure-B, a participant uses Firefox-B to perform a similar *Initial Visit* scenario and a *Revisit* scenario, but the password manager is CSF-BPM. The detailed tasks in these two procedures are listed in Table 3 and Table 4, respectively.

We provided these tasks to the participants for them to perform the two procedures. Before they perform the procedures, we also explained the main differences between the tasks in these two procedures. For example, we mentioned that the password manager in Firefox-A uses a recovery key and a master password to ensure the security; a user needs to supply the recovery key when the Sync mechanism is used and needs to supply the master password at least once in a browsing session. In contrast, the password manager in Firefox-B only uses a master password to ensure the security; it does not use a recovery key, but requires a user to wait for 10 seconds after supplying the master password once at the beginning of a browsing session. We also answered participants' questions on the usage of the two password managers.

We created the accounts of the three testing websites and the Firefox testing Sync account, so that there is no risk to the personal information or

Table 3: Tasks in Procedure-A using Firefox-A.

The Initial Visit Scenario:
A1: Open Firefox
A2: Go to the "Sync" tab, supply the testing Sync account and the recovery key
A3: Visit and log into mail.yahoo.com, www.amazon.com, and www.facebook.com, respectively
A4: Supply the testing master password once and let the password manager remember the accounts of the three testing websites
A5: Close Firefox
The Revisit Scenario:
A6: Repeat Tasks A1 to A2
A7: Revisit the three testing websites, supply the testing master password once, and log into the three websites after the password manager automatically fill the corresponding login forms
A8: Close Firefox

accounts of any participant. We also created the testing master password that is used in both procedures. To mitigate potential response bias, we randomly assigned one half of the participants to first perform Procedure-A and then Procedure-B, and assigned the other half of the participants to first perform Procedure-B and then Procedure-A.

6.3.3. Data Collection

We collected data through observation and questionnaire. When a participant was performing a procedure, we observed the progress of all the tasks. After a participant completed the two procedures, we asked the participant to answer a five-point Likert-scale (Strongly disagree, Disagree, Neither agree nor disagree, Agree, Strongly Agree) [74] questionnaire. The questionnaire consists of eight close-ended questions as listed in Table 5. We also asked participants to write down open-ended comments on using the password managers of Firefox-A and Firefox-B.

Participants were encouraged to ask us for a clarification of each individual question before providing the answer to it. Some participants indeed asked us for clarifications, so we can assume that those questions are clear

Table 4: Tasks in Procedure-B using Firefox-B.

The Initial Visit Scenario:
B1: Open Firefox
B2: Go to the "Security" tab, supply the testing Sync account
B3: On the same tab, supply the testing master password and wait for 10 seconds until the dialog box indicates a completion status
B4: Visit and log into mail.yahoo.com, www.amazon.com, and www.facebook.com, respectively
B5: Let the password manager remember the login accounts of the three testing websites
B6: Close Firefox
The Revisit Scenario:
B7: Repeat Steps B1 to B3
B8: Revisit the three testing websites and log into the three websites after the password manager automatically fill the corresponding login forms
B9: Close Firefox

to the participants. For one example, before answering Q1 and Q2, one participant asked us to further explain the reasons for using the recovery key in Firefox-A and having a 10-second waiting time in Firefox-B. We provided more details about these two mechanisms to the participant; we did not compare their security strengths and weaknesses, but mentioned that these two questions are only from the usability perspective. For another example, before answering Q3 and Q4, one participant asked us to explain the word "difference" in those two statements. We clarified that the difference refers to any perceivable difference from the user interaction perspective such as new dialog box, additional webpage, and different response time from browsers.

6.3.4. Results and Analysis

We observed that all the 30 participants successfully completed the two procedures. We converted the responses to the Likert-scale questionnaire to numeric values (1=Strongly disagree, 2=Disagree, 3=Neither agree nor disagree, 4=Agree, 5=Strongly Agree). Figure 8 illustrates the mean ratings to the eight questions. Strictly speaking, since the responses are ordinal data,

Table 5: The eight close-ended questions.

Q1: In Firefox-A, it is a burden to supply the recovery key every time after configure the testing Sync account
Q2: In Firefox-B, it is a burden to wait for 10 seconds every time before start my browsing
Q3: I cannot perceive any difference between Firefox-A and Firefox-B when they remember an online password in an initial visit
Q4: I cannot perceive any difference between Firefox-A and Firefox-B when they automatically fill a remembered online password in a revisit
Q5: Overall, it is easy to use the password manager of Firefox-A
Q6: Overall, it is easy to use the password manager of Firefox-B
Q7: Overall, I would like to use the password manager of Firefox-A in the future
Q8: Overall, I would like to use the password manager of Firefox-B in the future

they do not necessarily have interval scales. We performed such a conversion simply to ease the comparison of the responses from a relative perspective. In practice, this type of conversion is acceptable and commonly used such as in [39]. We mainly use t-tests (one-sample and two-sample) with 95% confidence interval to compare these mean ratings.

The mean rating to Q1 is 4.17. One-sample t-test against the test-value 4 shows this mean rating is higher than 4 *without* statistical significance (two-tailed p value is 0.134). This result indicates that most participants do *agree* that "In Firefox-A, it is a burden to supply the recovery key every time after configure the testing Sync account". In other words, supplying a 26-character recovery key in the password manager of the original Firefox is indeed a burden to most users. In contrast, the mean rating to Q2 is 2.17. One-sample t-test against the test-value 2 shows this mean rating is higher than 2 *without* statistical significance (two-tailed p value is 0.169). This result indicates that most participants *disagree* that "In Firefox-B, it is a burden to wait for 10 seconds every time before start my browsing". In other words, it is quite acceptable for users to wait for 10 seconds at the

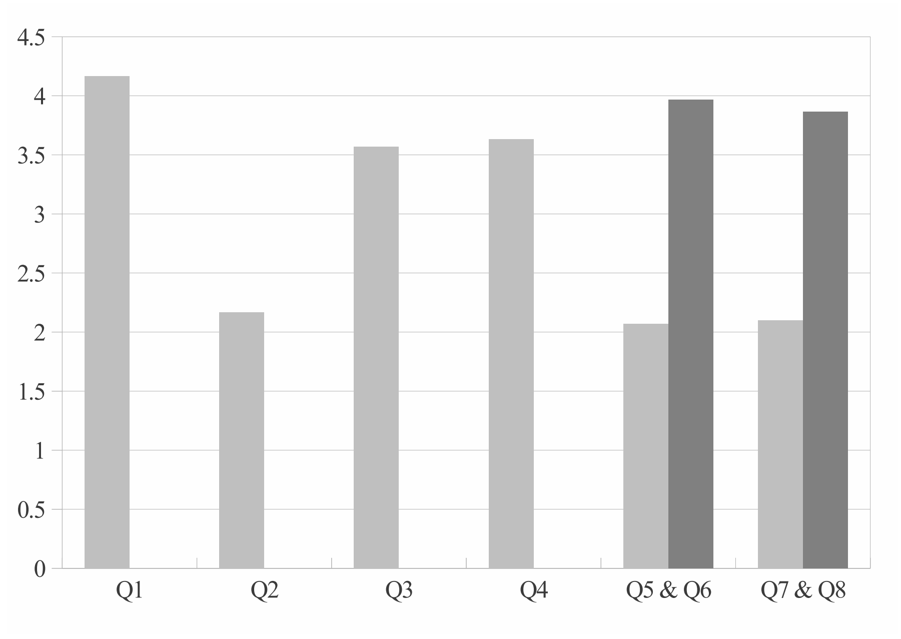


Figure 8: Mean ratings to questions Q1 to Q8.

beginning of a browsing session when they use CSF-BPM.

The mean ratings to Q3 and Q4 are 3.57 and 3.63, respectively. One-sample t-test against the test-value 3 shows both mean ratings are higher than 3 *with* statistical significance (two-tailed p values are less than 0.0001). One-sample t-test against the test-value 4 shows both mean ratings are lower than 4 *with* statistical significance (two-tailed p values are 0.0015 and 0.0028, respectively). One-sample t-test against the test-value 3.5 shows both mean ratings are higher than 3.5 *without* statistical significance (two-tailed p values are 0.595 and 0.245, respectively). These results indicate that most participants either *agree* or *neither agree nor disagree* that they cannot perceive the differences between Firefox-A and Firefox-B when the two browsers save an online password in an initial visit and automatically fill a saved password in a revisit. These results can be explained by the fact that in our implementation, the interfaces for triggering the remembering and autofill of passwords are not changed, and only the operations happening behind the scenes are changed (Section 5).

The mean rating to Q5 is 2.07. One-sample t-test against the test-value 2 shows this mean rating is higher than 2 *without* statistical significance (two-tailed p value is 0.424). The mean rating to Q6 is 3.97. One-sample t-test against the test-value 4 shows this mean rating is lower than 4 *without* statistical significance (two-tailed p value is 0.326). Meanwhile, two-sample t-test shows the mean rating to Q5 is lower than that to Q6 *with* statistical significance (two-tailed p value is less than 0.0001). These results clearly indicate that most participants *disagree* that "it is easy to use the password manager of Firefox-A", and *agree* that "it is easy to use the password manager of Firefox-B".

The mean rating to Q7 is 2.1. One-sample t-test against the test-value 2 shows this mean rating is higher than 2 *without* statistical significance (two-tailed p value is 0.415). The mean rating to Q8 is 3.87. One-sample t-test against the test-value 4 shows this mean rating is lower than 4 *without* statistical significance (two-tailed p value is 0.161). Meanwhile, two-sample t-test shows the mean rating to Q7 is lower than that to Q8 *with* statistical significance (two-tailed p value is less than 0.0001). These results clearly indicate that most participants would like to use CSF-BPM rather than the original password manager of Firefox in the future.

In our open-ended question, we asked participants to write down any other comments (if they have) regarding using the password managers of Firefox-A and Firefox-B. We found 22 (or 73.3% of) participants commented that supplying the 26-character recovery key is a burden to them. Their main opinion is that the recovery key is too long to be remembered or conveniently carried with, and they may make mistakes when they supply this recovery key. Some of them were even worried about losing the recovery key thus making the saved passwords irrecoverable. We also found nine participants commented that waiting for 10 seconds is acceptable especially for the sake of better security. These results further confirmed the difference in the participants' responses to Q1 and Q2, and further explained the difference in the participants' responses to Q5 and Q6. Our overall conclusion is that CSF-BPM does have usability advantages over the original password manager of Firefox.

7. Discussion

We analyzed in Section 4 that CSF-BPM provides a high level of security. We further evaluated in Section 6 the correctness, performance, and usability of our Firefox version CSF-BPM. We now briefly discuss a few main limitations of CSF-BPM.

First, if a CSF-BPM user forgets the SSMP, all the passwords saved on SRS services cannot be correctly decrypted. Therefore, remembering the SSMP becomes very important for CSF-BPM users. However, remembering an SSMP should be much easier than remembering many strong passwords for different websites.

Second, at the beginning of a Web browsing session, a user has to wait for 10 seconds so that CSF-BPM can complete the mainKey derivation. However, once the mainKey is derived, password remembering and autofill operations can be smoothly performed as usual.

Third, our current CSF-BPM is implemented in JavaScript. The security and performance of CSF-BPM can be further improved if those cryptographic algorithms are implemented in C++. For example, those cryptographic algorithms can be implemented into an XPCOM [70] component for Firefox using C++.

Fourth, we expect that the SSMP is strong with its strength [55, 56] assured by the traditional proactive password checking techniques and certain length requirements [57, 58, 59], or by the latest reactive proscriptive intervention techniques [60]. However, these techniques are statistical in nature and do not ensure an absolutely strong password for every single user. Therefore, insincere cloud storage service providers or attackers who can steal the encrypted data may still be able to launch brute force attacks on weak SSMPs. In addition, although the special UI component of CSF-BPM can help protect SSMP against phishing attacks (Section 4.2), users should still pay attention to any suspicious dialog box that asks for the SSMP.

Finally, in our threat model we assumed that it is very difficult for malware to directly identify cryptographic keys from a computer's memory and malware can be removed from the system by security-conscious users in a timely manner. Relatively speaking those assumptions are reasonable as justified in Section 3, but users should still pay attention to the potential risks. With a successful drive-by download attack and with the malware persisting on a user's computer, attackers may still log keystrokes and steal the data from the memory to obtain the master password, mainKey, and website passwords. Therefore, we expect users not to type the SSMP or log into a website if they perceive (e.g., with the help from the anti-malware programs on their computers) some suspicious activities; instead, they should immediately address the malware problem by either cleaning up or reinstalling the system. This is a common expectation for using all the browser-based password managers.

8. Conclusion

In this paper, we uncovered the vulnerabilities of existing BPMs and analyzed how they can be exploited by attackers to crack users' saved passwords. Moreover, we proposed a novel Cloud-based Storage-Free BPM (CSF-BPM) design to achieve a high level of security with the desired confidentiality, integrity, and availability properties. We implemented a CSF-BPM system and seamlessly integrated it into the Firefox Web browser. We evaluated the correctness, performance, and usability of this system. Our evaluation results and analysis demonstrate that CSF-BPM can be efficiently and conveniently used to manage online passwords. We believe CSF-BPM is a rational design that can also be integrated into other popular Web browsers to make the online experience of Web users more secure, convenient, and enjoyable.

Acknowledgement

The authors sincerely thank anonymous reviewers for their valuable suggestions and comments. This research was supported in part by the U.S. National Science Foundation under grant CNS-1359542.

References

[1] J. Bonneau, C. Herley, P. C. van Oorschot, F. Stajano, The quest to replace passwords: A framework for comparative evaluation of web authentication schemes, in: Proceedings of the IEEE Symposium on Security and Privacy, 2012, pp. 553–567.

[2] C. Herley, P. C. van Oorschot, A research agenda acknowledging the persistence of passwords, IEEE Security & Privacy 10 (1) (2012) 28–36.

[3] C. Herley, P. C. van Oorschot, A. S. Patrick, Passwords: If we're so smart, why are we still using them?, in: Proceedings of the Financial Cryptography, 2009, pp. 230–237.

[4] A. Adams, M. A. Sasse, Users are not the enemy, Commun. ACM 42 (12) (1999) 40–46.

[5] D. C. Feldmeier, P. R. Karn, Unix password security – ten years later, in: Proceedings of the Annual International Cryptology Conference (CRYPTO), 1989, pp. 44–63.

[6] S. Komanduri, R. Shay, P. G. Kelley, M. L. Mazurek, L. Bauer, N. Christin, L. F. Cranor, S. Egelman, Of passwords and people: Measuring the effect of password-composition policies, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI), 2011, pp. 2595–2604.

[7] R. Morris, K. Thompson, Password security: a case history, Commun. ACM 22 (11) (1979) 594–597.

[8] J. Yan, A. Blackwell, R. Anderson, A. Grant, Password memorability and security: Empirical results, IEEE Security and Privacy 2 (5) (2004) 25–31.

[9] M. Jakobsson, 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: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI), 2006, pp. 581–590.

[11] Anti-Phishing Working Group, `http://www.antiphishing.org`.

[12] D. Florêncio, C. Herley, A large-scale study of web password habits, in: Proceedings of the International Conference on World Wide Web (WWW), 2007, pp. 657–666.

[13] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. A. Kemmerer, C. Kruegel, G. Vigna, Your botnet is my botnet: analysis of a botnet takeover, in: Proceedings of the ACM Conference on Computer and Communications Security (CCS), 2009, pp. 635–647.

[14] A. Bessani, M. Correia, B. Quaresma, F. André, P. Sousa, Depsky: dependable and secure storage in a cloud-of-clouds, in: Proceedings of The European Conference on Computer Systems (EuroSys), 2011.

[15] K. D. Bowers, A. Juels, A. Oprea, Hail: a high-availability and integrity layer for cloud storage, in: Proceedings of the ACM Conference on Computer and Communications Security (CCS), 2009, pp. 187–198.

[16] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, M. Walfish, Depot: Cloud storage with minimal trust, ACM Trans. Comput. Syst. 29 (4).

[17] R. A. Popa, J. Lorch, D. Molnar, H. J. Wang, L. Zhuang, Enabling security in cloud storage slas with cloudproof, in: Proceedings of the USENIX Annual Technical Conference, 2011.

[18] C. Wang, Q. Wang, K. Ren, N. Cao, W. Lou, Toward secure and dependable storage services in cloud computing, IEEE Trans. Serv. Comput. 5 (2) (2012) 220–232.

[19] Windows Azure Storage Team, Windows azure storage: a highly available cloud storage service with strong consistency, in: Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), 2011.

[20] A. Shamir, How to share a secret, Commun. ACM 22 (11) (1979) 612–613.

[21] M. Wu, R. C. Miller, G. Little, Web wallet: preventing phishing attacks by revealing user intentions, in: Proceedings of the Symposium on Usable Privacy and Security (SOUPS), 2006, pp. 102–113.

[22] 1Password., `https://agilebits.com/onepassword`.

[23] RoboForm Password Manager., `http://www.roboform.com/`.

[24] D. P. Kormann, A. D. Rubin, Risks of the passport single signon protocol, Comput. Networks 33 (1-6) (2000) 51–58.

[25] S.-T. Sun, Y. Boshmaf, K. Hawkey, K. Beznosov, A billion keys, but few locks: the crisis of web single sign-on, in: Proceedings of the New security Paradigms Workshop (NSPW), 2010, pp. 61–72.

[26] OpenID Authentication 2.0 - Final, `http://openid.net/specs/openid-authentication-2_0.html`.

[27] The OAuth 2.0 Authorization Framework, `http://tools.ietf.org/html/rfc6749`.

[28] D. Davis, F. Monrose, M. K. Reiter, On user choice in graphical password schemes, in: Proceedings of the USENIX Security Symposium, 2004, pp. 151–164.

[29] J. Thorpe, P. van Oorschot, Human-seeded attacks and exploiting hotspots in graphical passwords, in: Proceedings of the USENIX Security Symposium, 2007, pp. 103–118.

[30] J. Thorpe, P. C. van Oorschot, Towards secure design choices for implementing graphical passwords, in: Proceedings of the Annual Computer Security Applications Conference (ACSAC), 2004, pp. 50–60.

[31] J. A. Halderman, B. Waters, E. W. Felten, A convenient method for securely managing passwords, in: Proceedings of the International Conference on World Wide Web (WWW), 2005, pp. 471–479.

[32] B. Ross, C. Jackson, N. Miyake, D. Boneh, J. C. Mitchell, Stronger password authentication using browser extensions, in: Proceedings of the USENIX Security Symposium, 2005, pp. 17–32.

[33] K.-P. Yee, K. Sitaker, Passpet: convenient password management and phishing protection, in: Proceedings of the Symposium on Usable Privacy and Security (SOUPS), 2006, pp. 32–43.

[34] LastPass Password Manager., `https://lastpass.com/`.

[35] R. Zhao, C. Yue, K. Sun, Vulnerability and risk analysis of two commercial browser and cloud based password managers, ASE Science Journal 1 (4) (2013) 1–15.

[36] S.-T. Sun, K. Beznosov, The devil is in the (implementation) details: an empirical analysis of oauth sso systems, in: Proceedings of the ACM Conference on Computer and Communications Security (CCS), 2012.

[37] R. Wang, S. Chen, X. Wang, Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services, in: Proceedings of the IEEE Symposium on Security and Privacy, 2012.

[38] C. Yue, The devil is phishing: Rethinking web single sign-on systems security, in: Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET), 2013.

[39] S. Chiasson, P. C. van Oorschot, R. Biddle, A usability study and critique of two password managers, in: Proceedings of the USENIX Security Symposium, 2006, pp. 1–16.

[40] R. Zhao, C. Yue, All your browser-saved passwords could belong to us: A security analysis and a cloud-based new design, in: Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY), 2013.

[41] P. Bowen, J. Hash, M. Wilson, Information security handbook: A guide for managers, in: NIST Special Publication 800-100, 2007, `http://csrc.nist.gov/publications/nistpubs/800-100/SP800-100-Mar07-2007.pdf`.

[42] M. Cova, C. Kruegel, G. Vigna, Detection and analysis of drive-by-download attacks and malicious javascript code, in: Proceedings of the International Conference on World Wide Web (WWW), 2010, pp. 281–290.

[43] L. Lu, V. Yegneswaran, P. Porras, W. Lee, Blade: an attack-agnostic approach for preventing drive-by malware infections, in: Proceedings of the ACM Conference on Computer and Communications Security (CCS), 2010.

[44] A. Moshchuk, T. Bragin, S. D. Gribble, H. M. Levy, A crawler-based study of spyware in the web., in: Proceedings of the Annual Network & Distributed System Security Symposium (NDSS), 2006.

[45] N. Provos, P. Mavrommatis, M. A. Rajab, F. Monrose, All your iframes point to us, in: Proceedings of the USENIX Security Symposium, 2008.

[46] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, S. T. King, Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities., in: Proceedings of the Annual Network & Distributed System Security Symposium (NDSS), 2006.

[47] M. T. Louw, J. S. Lim, V. N. Venkatakrishnan, Enhancing web browser security against malware extensions, Journal in Computer Virology 4 (3) (2008) 179–195.

[48] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, E. W. Felten, Lest we remember: Cold boot attacks on encryption keys, in: Proceedings of USENIX Security Symposium, 2008.

[49] F. Hsu, H. Chen, T. Ristenpart, J. Li, Z. Su, Back to the future: A framework for automatic malware removal and system repair, in: Proceedings of the Annual Computer Security Applications Conference (ACSAC), 2006, pp. 257–268.

[50] E. Grosse, M. Upadhyay, Authentication at scale, IEEE Security and Privacy 11 (2013) 15–22.

[51] SQLite Home Page, `http://www.sqlite.org`.

[52] Windows CryptProtectData function, `http://msdn.microsoft.com/en-us/library/windows/desktop/aa380261(v=vs.85).aspx`.

[53] Windows CryptUnprotectData function, `http://msdn.microsoft.com/en-us/library/windows/desktop/aa380882(v=vs.85).aspx`.

[54] C. Bravo-Lillo, L. Cranor, J. Downs, S. Komanduri, S. Schechter, M. Sleeper, Operating system framed in case of mistaken identity: measuring the success of web-based spoofing attacks on os password-entry dialogs, in: Proceedings of the ACM Conference on Computer and Communications Security (CCS), 2012, pp. 365–377.

[55] W. E. Burr, D. F. Dodson, E. M. Newton, R. A. Perlner, W. T. Polk, S. Gupta, E. A. Nabbus, Electronic authentication guideline, in: NIST Special Publication 800-63-1, 2011, `http://csrc.nist.gov/publications/nistpubs/800-63-1/SP-800-63-1.pdf`.

[56] L. S. Clair, L. Johansen, W. Enck, M. Pirretti, P. Traynor, P. McDaniel, T. Jaeger, Password exhaustion: predicting the end of password usefulness, in: Proceedings of the International Conference on Information Systems Security, 2006, pp. 37–55.

[57] M. Bishop, D. V. Klein, Improving system security via proactive password checking, Computers & Security 14 (3) (1995) 233–249.

[58] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, J. Lopez, Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms, in: Proceedings of the IEEE Symposium on Security and Privacy, 2012, pp. 523–537.

[59] J. J. Yan, A note on proactive password checking, in: Proceedings of the New security Paradigms Workshop (NSPW), 2001, pp. 127–135.

[60] C. Herley, S. Schechter, Breaking our password hash habit – why the sharing of users' password choices for defensive analysis is an underprovisioned social good, and what we can do to encourage it, in: Proceedings of the Workshop on the Economics of Information Security (WEIS), 2013.

[61] B. Laurie, Nigori: Storing Secrets in the Cloud, `http://www.links.org/files/nigori-overview.pdf`.

[62] B. Kaliski, RFC 2898, PKCS5: Password-based cryptography specification version 2.0, `http://www.ietf.org/rfc/rfc2898.txt` (1999).

[63] Advanced encryption standard (AES), in: NIST FIPS 197, 2001, `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[64] M. Dworkin, Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality, in: NIST Special Publication 800-38C, 2004, `http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C.pdf`.

[65] X. Boyen, Halting password puzzles: hard-to-break encryption from human-memorable keys, in: Proceedings of the USENIX Security Symposium, 2007, pp. 119–134.

[66] W. Stallings, Cryptography and Network Security: Principles and Practice (6th Edition), Prentice Hall Press, 2013.

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

[68] Firefox Sync Service, `https://wiki.mozilla.org/Services/Sync`.

[69] T. Wu, The secure remote password protocol, in: Proceedings of the Annual Network & Distributed System Security Symposium (NDSS), 1998.

[70] XPCOM: Cross Platform Component Object Model, `https://developer.mozilla.org/en/XPCOM`.

[71] E. Stark, M. Hamburg, D. Boneh, Symmetric cryptography in javascript, in: Proceedings of the Annual Computer Security Applications Conference (ACSAC), 2009, pp. 373–381.

[72] R. Canetti, S. Halevi, M. Steiner, Mitigating dictionary attacks on password-protected local storage, in: Proceedings of the Annual International Cryptology Conference (CRYPTO), 2006, pp. 160–179.

[73] A. Whitten, J. D. Tygar, Why Johnny can't encrypt: a usability evaluation of PGP 5.0, in: Proceedings of the USENIX Security Symposium, 1999.

[74] Likert scale., `http://en.wikipedia.org/wiki/Likert_scale`.