Toward Secure and Convenient Browsing Data Management in the Cloud

Chuan Yue (cyue@uccs.edu), *University of Colorado Colorado Springs, USA*

Abstract

Cloud and Web-centric computing is a significant trend in computing. However, the design and development of modern Web browsers failed to catch up this significant trend to address many challenging Browsing Data Insecurity and Inconvenience (referred to as BDII) problems that bother millions of Web users. In this position paper, we present our preliminary investigation on the BDII problems of the five most popular Web browsers and highlight the necessity and importance of addressing those problems. We also propose to explore a novel Cloud computing Age Browser (referred to as CAB) architecture that leverages the reliability and accessibility advantages of cloud storage services to fundamentally address the BDII problems.

***K*eywords:** Browser, Cloud, Security, Convenience

1 Introduction

Cloud and Web-centric computing is a significant trend in computing. However, the design and development of modern Web browsers failed to catch up this significant trend in computing to address many challenging Browsing Data Insecurity and Inconvenience (referred to as BDII) problems that bother millions of Web users.

Each set of `browsing data` includes a complete collection of different types of data associated with each particular browser profile. Those different types of data can be classified into three categories: (1) user-saved data such as website passwords, bookmarks, form autofill values, and input autocomplete values; (2) browser-saved data such as browsing history and HTTP cookies; (3) browser preference settings such as homepage, language, appearance, security, and privacy that are either related or not related to the first two categories of data.

Ideally, users' browsing data should be securely protected and should be conveniently accessible. By *securely protected*, we mean the confidentiality (including privacy), integrity, and availability of users' browsing data are assured. By *conveniently accessible*, we mean the complete set of browsing data for each browser profile is consistently maintained, is highly browser-agnostic, and is available and readily usable anytime, anyplace, and on any computer. Unfortunately, the reality is that the challenging BDII problems have never been seriously addressed by either the vendors of modern Web browsers or researchers.

In this position paper, we make two main contributions. One is that in Section 2, we present our preliminary investigation on the BDII problems of the five most popular browsers and highlight the necessity and importance of addressing those problems. The other is that in Section 3, we propose to explore a novel Cloud computing Age Browser (referred to as CAB) architecture to fundamentally address the BDII problems.

CAB does not save any browsing data on a user's computer – all the browsing data will be protected and completely stored in the cloud. It aims to address the BDII problems by properly leveraging the reliability and accessibility advantages of cloud storage services while overcoming the potential concerns of using those services. The CAB architecture can also bring one additional but significant security benefit – it can reduce the attack surfaces of modern browsers by minimizing the privileges for file operations, which are often the root causes of many Web-based attacks. We review related work in Section 4, and discuss the adoption incentives, the deployment and development strategy, and implementation challenges for CAB in Section 5.

2 BDII Problems

Table 1 lists the basic information about the storage of five selected types of browsing data on the latest versions of the five most popular browsers on Windows 7. Other types of browsing data mentioned in Section 1 are not listed due to space limitation. We can see that these

Table 1: Storage of five selected types of browsing data on the five most popular Web browsers.

Browser	Website passwords	Bookmarks	HTTP cookies	Browsing history	Preference settings
Internet Explorer (9.0)	Windows registry	Windows shortcut files	text files	Windows shortcut files	Windows registry
Firefox (17.0)	SQLite database	SQLite database	SQLite database	SQLite database	JavaScript file
Google Chrome (23.0)	SQLite database	text file	SQLite database	SQLite database	text file
Safari (5.1.7)	property list file	property list file	binary file	property list file	property list file
Opera (12.11)	binary file	text file	encoded text file	text file	text file

browsers persist browsing data to the disk using various file types such as SQLite database, Windows registry, Windows shortcut file, text file, encoded text file, property list file, JavaScript file, and binary file.

Our preliminary investigation shows many BDII problems exist in the five most popular browsers. Basically, these browsers do not provide a strong security protection to users' browsing data, and do not provide sufficient support to ensure the consistency and compatibility of a user's complete set of browsing data across different computers and across different browser products.

2.1 Browsing data insecurity problems

"Where a threat intersects with a vulnerability, risk is present" (NIST SP800-100). For browsing data, the threat sources are attackers who want to steal users' sensitive information. The basic *threat model* we consider throughout this paper is that attackers can temporarily (e.g., in a few seconds) install malware on a user's computer using popular attacks such as drive-by downloads [4, 12, 14, 16, 20]. The installed malware can then steal the browsing data stored on the disk.

Such threats are prevalent and have high impacts because browsing data often contain very sensitive information. For example, website passwords have been continuously targeted by various cracking and harvesting attacks [11, 18, 25]; HTTP cookie stealing attacks can cause severe security and privacy breaches [6, 26]; browsing history sniffing attacks can also cause severe security and privacy breaches [21, 23]. Therefore, we do not intend to further identify threat sources, but focus on highlighting the vulnerabilities that can be exploited by threat sources to easily steal sensitive browsing data.

We assume malware will not persist on the victim's machine – anti-malware software such as Microsoft Forefront Endpoint Protection may eventually detect and remove the malware, or solutions such as the Back to the Future framework [9] may restore the system to a prior good state and preserve the system integrity. This is a reasonable assumption that is also made in other systems such as Google's two-step verification system [8].

One particular type of browsing data is the login account usernames and passwords for different websites. A user can allow the password manager of a browser to save the login account information and later automatically fill the login forms on behalf of the user. In all the five most popular Web browsers, the protection to the saved website passwords is very weak – the encrypted passwords stored by the latest versions of those browsers could be trivially decrypted by attackers for them to log into victims' accounts on the corresponding websites. We refer readers to our recent paper [27] for more details.

Even worse, all the other types of browsing data are not protected at all by any of those browsers. Given the rampant of Web-based attacks over the Internet [17] and the aforementioned attack examples [4, 6, 11, 12, 14, 16, 18, 20, 21, 23, 25, 26], data confidentiality, integrity, and availability can be easily compromised and many security and privacy breaches such as identity theft and behavior tracking can occur if those unprotected browsing data can be accessed by attackers or unauthorized parties.

2.2 Browsing data inconvenience problems

To be conveniently accessible, the complete set of browsing data for each browser profile should be consistently maintained, highly browser-agnostic, available and readily usable anytime, anyplace, and on any computer. However, none of the existing popular browsers can provide such a level of convenience.

Browsers such as Firefox, Google Chrome, and Opera only provide a limited data consistency support with a synchronization feature that can synchronize partial browsing data to their own cloud storage servers and among users' computers. Because multiple copies of browsing data for the same browser profile could exist on different computers and each copy may contain some most recent data, synchronization conflicts become very complex and may not be properly resolved in these browsers. Browsers such as Internet Explorer and Safari do not provide a built-in synchronization feature.

Furthermore, the compatibility of browsing data among these five browsers is very poor. The file types of the persisted browsing data vary from browser to browser as shown in Table 1. The structures of those files for the same type of browsing data also vary from browser to browser. Google Chrome and Firefox have limited capabilities to import certain browsing data from other browsers. But overall, sharing browsing data across dif-

ferent browser products is poorly supported; synchronizing the same set of browsing data across different browser products is much more challenging and has never been achieved by browser vendors or researchers.

In realistic scenarios, however, it would be very beneficial to users if most types of browsing data such as the saved website passwords and preference settings could be browser-agnostic. For example, when a user temporarily works on a library computer that only has Internet Explorer installed, the user should be able to conveniently use the same browser profile that is mainly used on the Firefox of his or her office computer. For another example, when a zero-day vulnerability on a user's default browser such as Opera is disclosed but the security patch is not released yet, the user should be able to choose another browser such as Google Chrome to conveniently use the same browser profile.

Based on these preliminary investigation results, we believe it is necessary and important to thoroughly investigate the BDII problems, to comprehensively identify the essential requirements for addressing the BDII problems, and to rationally explore new browser architectures that could best address the BDII problems of modern browsers.

3 Design of the CAB Architecture

We consider the following six requirements listed in Table 2 as the candidates of essential requirements for addressing BDII problems. This list mainly includes the requirements that are not or only partially met by modern Web browsers, but are essential and critical based on our preliminary investigation results in Section 2.

Table 2: Requirements for addressing BDII problems.

Requirements	Current Status and Justification of Necessity
REQ1: provide a strong protection to ensure the confidentiality, integrity, and availability of all types of browsing data	weak protection to passwords, no protection at all to other browsing data in existing browsers
REQ2: ensure the consistency of the complete set of browsing data for each browser profile across different computers	partial synchronization in existing browsers with complex/improper conflicts resolution
REQ3: support the sharing and synchronization of the complete set of browsing data across different browser products	poor browsing data compatibility, very difficult to even simply share browsing data across different browser products
REQ4: assure the complete set of browsing data be available and readily usable anytime, anyplace, and on any computer	only partial browsing data are accessible by some browsers from their own dedicated servers
REQ5: offer good usability with minimum user intervention	too much manual effort (e.g., import/export) in existing browsers
REQ6: follow the principle of least privilege [19]	too many file operations in existing browsers

3.1 Design overview

We propose to explore a novel CAB (Cloud computing Age Browser) architecture that could best address the BDII problems by taking advantage of the exciting opportunities brought by the advances in cloud computing and especially in cloud storage services (Section 4).

The most distinctive feature of the proposed CAB architecture is that all the browsing data will be protected and completely stored in the cloud – nothing needs to be stored on a user's computer. We want to move the storage into the cloud so that the essential requirements REQ2 and REQ4 (Table 2) could be achieved by properly leveraging the reliability and accessibility advantages of cloud storage services (Section 4). In the long run, trustworthy cloud storage services [1, 3, 13, 15, 22] could even better protect regular users' browsing data than local computers (which may not be timely and properly patched) do.

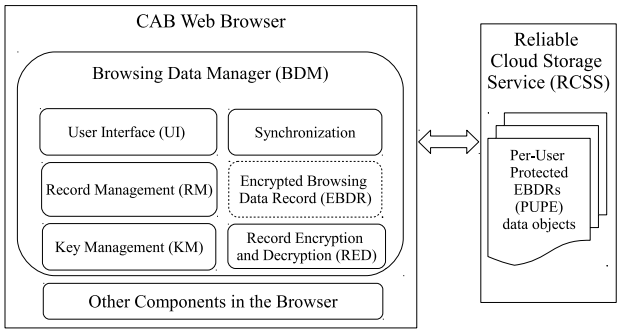

Figure 1: The high-level design of the CAB architecture.

Figure 1 illustrates the tentative high-level design of CAB. The essential part of this design is a simple and generic Browsing Data Manager (BDM) component that could be easily and seamlessly integrated into existing popular browsers to enable them to adopt the CAB architecture. BDM consists of five subcomponents: UI, RM, RED, KM, and Synchronization.

UI provides simple configuration and management user interfaces accessible at a single location. BDM does not include any traditional persistent storage such as a file or a database; instead, it will use an RM subcomponent to replace the traditional persistent storage of any browser. RM maintains an in-memory array of EBDRs (Encrypted Browsing Data Records) using a corresponding interface for adding, deleting, modifying, and searching EBDRs. RED performs browsing data encryption and decryption operations. KM performs cryptographic key management operations. The synchronization subcomponent will transparently retrieve (or upload) a PUPE (Per-User Protected EBDRs) data object from (or to) an RCSS (Reliable Cloud Storage Service) in real-time whenever needed.

An RCSS simply needs to support user authentication over HTTPS and per-user data storage, which are basic functionalities provided by most of the cloud storage services (Section 4). It will store a PUPE data object for each RCSS user. It simply needs to be a reliable cloud storage service and it does not need to provide any special computational support to CAB. The communication protocol between CAB and an RCSS is also very simple: after a user authenticates to the RCSS, the synchronization subcomponent of BDM will transparently send HTTPS requests to the RCSS (e.g., using the REST APIs to be mentioned in Section 4.2) to retrieve or upload the PUPE data object of the user.

To use CAB, a user needs to remember a Single Strong Master Password (referred to as SSMP) with the strength assured by a proactive password checker and certain length requirement [2, 11, 24]. SSMP will be used to derive keys for protecting all the browsing data. The user also needs to set up an RCSS account and configure the URL address of RCSS once through the UI subcomponent. At the beginning of each browsing session, the user needs to authenticate to the RCSS and provide the SSMP to BDM; these two steps need to be performed only once in each browsing session through the UI subcomponent, thus reducing the risk of phishing attacks against the RCSS account and SSMP. After that, BDM will transparently take care of everything else related to the browsing data management.

3.2 EBDR and PUPE

The EBDR (Encrypted Browsing Data Record) and PUPE (Per-User Protected EBDRs) data structures must be compatible across different browsers and must be extensible for future need, so that the essential requirement REQ3 (Table 2) could be achieved.

We plan to define two types of EBDRs: profile level EBDR and website level EBDR. A profile level EBDR will contain profile specific information such as encrypted preference settings or the encrypted browsing history, and it needs to be decrypted as soon as a user successfully authenticates to the RCSS and provides the SSMP to the BDM component of CAB. A website level EBDR will contain website specific information such as the encrypted password for a website or the encrypted HTTP cookies for a website, and it only needs to be decrypted when a user visits a particular website.

The structure of a PUPE object should be extensible and relatively simple. It will contain the *protectedEBDRs* and all the information related to the protection algorithms and parameters. The protectedEBDRs is the protected result of the entire concatenated EBDRs of an RCSS user. Each PUPE data object can be simply saved as a binary or encoded string object for an RCSS user because its structure does not need to be known or taken care of by any RCSS. Such a PUPE data object design makes the selection of protection algorithms and the selection of RCSS services very flexible.

3.3 Data protection mechanism

We plan to achieve the essential requirement REQ1 (Table 2) and provide a high level of security guarantee by: (1) mandating the SSMP with the strength assured by a proactive password checker and certain length requirement [2, 11, 24], (2) using a strong key derivation function (such as PBKDF2 – Password-Based Key Derivation Function Version 2 defined in the PKCS5 specification [10]) with randomly generated salts and large iteration counts, and (3) employing NIST-approved authenticated encryption algorithms (such as CCM – Counter with CBC-MAC [5]) to simultaneously enforce strong confidentiality and authenticity (integrity) on the PUPE data object of each RCSS user.

All the computations including salt generation, key derivation, encryption, and decryption etc. are performed by the BDM component of CAB. Neither the SSMP nor any derived cryptographic key will be revealed to an RCSS or a third party. Therefore, even if attackers (including insiders of an RCSS) can steal the saved PUPE data object, it is computationally infeasible for attackers to decrypt the stolen PUPE data object to obtain a user's browsing data.

This mechanism is usable (REQ5 in Table 2) because it purely uses password-based key derivation techniques and it only changes the cryptographic operations happening behind the scenes of browsers. This mechanism is also very flexible. Whenever necessary, the BDM component of CAB can transparently change the protection algorithms and their corresponding parameters. A user also has the flexibility to change SSMP whenever necessary. In these cases, all what need to be done by BDM is to simply update the PUPE data object, and upload the new PUPE data object to the RCSS.

3.4 Least privilege

The rampant of Web-based attacks [17] can, to a large extent, be attributed to the vulnerabilities in Web browsers such as logic flaws or bugs, violation of the principle of least privilege [19], and weak isolation among components or origins. Those vulnerabilities can be exploited by attacks such as drive-by downloads [4, 12, 14, 16, 20] to install malware on a computer without user approval.

By moving the storage into the cloud, CAB can perform browsing data management without reading, writing, or manipulating local files and directories (Table 1).

In other words, CAB can completely remove the necessity of using the privileges for file operations in supporting browsing data management – it only needs to use those privileges in supporting user-initiated operations such as explictly saving or uploading files. Minimizing the privileges for file operations (REQ6 in Table 2) can reduce the attack surfaces and reduce the risk of drive-by downloads attacks, in which storing files to a local file system is an essential phase. This could be a significant security benefit brought by CAB to modern browsers.

4 Related Work

We review cloud-based browsers to highlight CAB is completely different from them. We review cloud storage services to justify why we will build CAB upon them.

4.1 Existing cloud-based Web browsers

We review four popular cloud-based browser products from industry: Amazon Silk Browser, Cloud Browser [28], Opera Mini, and Puffin Browser [29]. In essence, all these cloud-based browsers follow a split-browsing approach with the goal of improving the Web browsing performance on mobile handheld devices. They use extra servers in the cloud to relay the HTTP requests and responses between mobile handheld devices and remote websites. They improve the mobile browsing performance mainly by using those servers in the cloud to perform the time-consuming webpage rendering and even JavaScript interpretation tasks.

CAB is completely different from those cloud-based browsers. First, the objectives are different. Those browsers improve the performance of mobile browsing, while CAB addresses the challenging BDII problems of modern browsers. Second, the security foundations are different. Those browsers rely on cloud servers to perform sensitive computations. Computing on encrypted data is possible [7], but it is still far away from being practical for cloud servers to render sensitive webpages without decrypting them in the first place. In contrast, CAB has a solid security foundation because (1) it simply uses the storage services provided by cloud servers without requiring any special computational support, and (2) it only saves authenticated and encrypted browsing data in the cloud. Third, those browsers target at mobile devices, but CAB explores a generic design that is applicable to both desktop computers and mobile devices.

4.2 Existing cloud storage services

Many cloud storage services such as Amazon Cloud Drive, Dropbox, Google Drive, HP Cloud Object Storage, iCloud, and Microsoft SkyDrive have been deployed and widely used. Many of these and other cloud storage services offer free accounts and storage spaces to regular users. Most of them follow the predominant REST (Representational State Transfer) Web service design model to allow different client applications to easily access them. Recent research advances further demonstrate the continuous improvements in the reliability, accessibility, and security of cloud storage services [1, 3, 13, 15, 22]. We believe this trend will continue with the joint effort from both industry and academia.

We do not propose to build any new cloud storage service, but focus on enabling our CAB to take advantage of these free, widely deployed, and easily accessible cloud storage services to address the BDII problems.

5 Discussions

We now take a top-down approach to further discuss the adoption incentives, the deployment and development strategy, and some implementation challenges for CAB.

5.1 Adoption incentives

By meeting the essential requirements listed in Table 2, CAB will bring significant security and convenience benefits to Web users as highlighted in the previous sections; this will be the most important incentive for both users and browser vendors to adopt CAB. For browser vendors, another important incentive lies in the potential of easy-and-smooth integration of CAB due to its simple and generic design as highlighted in Section 3.1. This potential will be further explored in the deployment and development of CAB.

5.2 Deployment and development strategy

To make the integration of CAB to popular Web browsers as easy and smooth as possible, we plan to first build and deploy CAB as browser extensions that can be seamlessly installed and used on Firefox and Chromium [30]. These browser extensions can directly meet the essential requirements from REQ1 to REQ5 in Table 2, and can be incrementally deployed on users' browsers. These browser extensions will also be easily configurable to directly use most of the existing cloud storage services (Section 4.2) without requiring any modification to them.

Next, we will modify the source code of the two browsers to minimize the privileges for file operations and further meet the REQ6 in Table 2.

Finally, after extensively evaluating the correctness, performance, consistency, compatibility, convenience, and security of the CAB browser extensions and the modified browsers, we will suggest vendors to integrate CAB into the browsers as a built-in feature.

5.3 Implementation challenges

We will carefully consider a number of detailed challenges in the implementation of CAB. Here are some examples: the interfaces between the Browsing Data Manager (Figure 1) and other components in the browser should not be too complex; the granularity of the EBDRs (Section 3.2) as well as the synchronization frequency of the PUPE object should be properly determined based on browsing data usage characteristics; a good balance between security and usability should be achieved in the data protection mechanism (Section 3.3); configurability and flexibility should be considered in the implementation to address the computational and user interface differences between desktop computers and mobile devices.

6 Conclusion

In this position paper, we highlighted the necessity and importance of addressing the BDII (Browsing Data Insecurity and Inconvenience) problems of modern Web browsers. We also proposed a CAB (Cloud computing Age Browser) architecture to fundamentally address the BDII problems. We sincerely welcome your suggestions and expect to further discuss the challenges in addressing the BDII problems and realizing the CAB architecture.

7 Acknowledgments

The author thanks anonymous reviewers for their insightful comments and valuable suggestions, thanks Mr. Rui Zhao for helping verify the information presented in Table 1, and thanks Drs. Chunqiang Tang, Byung Chul Tak, and Ashish Kundu for their valuable feedback and discussion about this paper. This work was partially supported by a UCCS 2012-2013 CRCW research grant.

References

[1] BESSANI, A., CORREIA, M., QUARESMA, B., ANDRÉ, F., AND SOUSA, P. Depsky: dependable and secure storage in a cloud-of-clouds. In *Proc. of EuroSys* (2011).

[2] BISHOP, M., AND KLEIN, D. V. Improving system security via proactive password checking. *Computers & Security 14*, 3 (1995).

[3] BOWERS, K. D., JUELS, A., AND OPREA, A. Hail: a high-availability and integrity layer for cloud storage. In *Proc. of CCS* (2009).

[4] COVA, M., KRUEGEL, C., AND VIGNA, G. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proc. of WWW* (2010).

[5] DWORKIN, M. Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. In *NIST Special Publication 800-38C* (2004).

[6] FU, K., SIT, E., SMITH, K., AND FEAMSTER, N. Do's and donts of client authentication on the web. In *Proc. of USENIX Security Symposium* (2001).

[7] GENTRY, C. Computing Arbitrary Functions of Encrypted Data. *Commun. ACM 53*, 3 (2010).

[8] GROSSE, E., AND UPADHYAY, M. Authentication at scale. *IEEE Security and Privacy 11* (2013), 15–22.

[9] HSU, F., CHEN, H., RISTENPART, T., LI, J., AND SU, Z. Back to the future: A framework for automatic malware removal and system repair. In *Proc. of ACSAC* (2006).

[10] KALISKI, B. RFC 2898, PKCS5: Password-Based Cryptography Specification Version 2.0, 1999.

[11] KELLEY, P. G., ET AL. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *Proc. of IEEE Symposium on S&P* (2012).

[12] LU, L., YEGNESWARAN, V., PORRAS, P., AND LEE, W. Blade: an attack-agnostic approach for preventing drive-by malware infections. In *Proc. of CCS* (2010).

[13] MAHAJAN, P., SETTY, S., LEE, S., CLEMENT, A., ALVISI, L., DAHLIN, M., AND WALFISH, M. Depot: Cloud storage with minimal trust. *ACM Trans. Comput. Syst. 29*, 4 (2011).

[14] MOSHCHUK, A., BRAGIN, T., GRIBBLE, S. D., AND LEVY, H. M. A crawler-based study of spyware in the web. In *Proc. of NDSS* (2006).

[15] POPA, R. A., LORCH, J., MOLNAR, D., WANG, H. J., AND ZHUANG, L. Enabling security in cloud storage slas with cloudproof. In *Proc. of USENIX Annual Technical Conference* (2011).

[16] PROVOS, N., MAVROMMATIS, P., RAJAB, M. A., AND MONROSE, F. All your iframes point to us. In *Proc. of USENIX Security Symposium* (2008).

[17] PROVOS, N., RAJAB, M. A., AND MAVROMMATIS, P. Cybercrime 2.0: when the cloud turns dark. *Commun. ACM 52*, 4 (2009).

[18] RACHNA DHAMIJA, J.D.TYGAR, AND MARTI HEARST. Why phishing works. In *Proc. of CHI* (2006).

[19] SALTZER, J. H., AND SCHROEDER, M. D. The protection of information in computer systems. *Proc. of IEEE 63*, 9 (1975).

[20] WANG, Y.-M., BECK, D., JIANG, X., ROUSSEV, R., VERBOWSKI, C., CHEN, S., AND KING, S. T. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *Proc. of NDSS* (2006).

[21] WEINBERG, Z., CHEN, E. Y., JAYARAMAN, P. R., AND JACKSON, C. I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks. In *Proc. of IEEE Symposium on S&P* (2011).

[22] WINDOWS AZURE STORAGE TEAM. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proc. of ACM SOSP* (2011).

[23] WONDRACEK, G., HOLZ, T., KIRDA, E., AND KRUEGEL, C. A practical attack to de-anonymize social network users. In *Proc. of IEEE Symposium on S&P* (2010).

[24] YAN, J. J. A note on proactive password checking. In *Proc. of New security Paradigms Workshop (NSPW)* (2001).

[25] YUE, C. Preventing the revealing of online passwords to inappropriate websites with LoginInspector. In *Proc. of USENIX LISA* (2012).

[26] YUE, C., XIE, M., AND WANG, H. An Automatic HTTP Cookie Management System. *Journal of Computer Networks (COMNET), Elsevier 54*, 13 (2010).

[27] ZHAO, R., AND YUE, C. All your browser-saved passwords could belong to us: A security analysis and a cloud-based new design. In *Proc. of ACM CODASPY* (2013).

[28] Cloud Browser. `http://www.alwaysontechnologies.com/cloudbrowse/features/`.

[29] Puffin Browser. `http://www.puffinbrowser.com/`.

[30] The Chromium Projects. `http://www.chromium.org/`.