Industry-Wide Misunderstandings of HTTPS

Stephen Bono(✉) and Jacob Thompson(✉)

Independent Security Evaluators, Baltimore, USA
{sbono, jthompson}@securityevaluators.com

Abstract. In a survey of 30 sites that serve sensitive content over an HTTPS-protected connection, we found that over 70 % of them failed to appropriately prevent disk caching, and left unencrypted sensitive content behind on end-users' machines, at risk for later exposure. Moreover, over half of the sites that failed to prevent disk caching appeared to have attempted to do so using outdated, non-standard, or erroneous methods, some of which failed entirely, while others were only successful at preventing disk caching in certain browsers, but not all.

In an effort to explain this wide-spread failure, our research has uncovered drastically inconsistent behavior across browsers, inconsistent support of standard and non-standard anti-disk caching directives, and even inconsistent and incorrect recommendations from authoritative sources in the security community. Through this history we show that web developers are not solely to blame, and that web browser developers, web server developers, security professionals and authors of online sources, and perhaps even the standards bodies should share in this failure.

In this paper, we identify the disk caching behaviors of all major browsers, and describe how to reliably prevent disk caching for each of them. We present the results of our site survey, demonstrating wide-spread failures to prevent disk caching of sensitive data. We introduce a tool for Firefox users to reliably prevent disk caching of HTTPS protected content, despite failures by the web application, and we provide an online tool to help web developers identify how to reliably prevent disk caching across multiple browsers. Lastly, we make recommendations to the various parties with a hand in this failure on how to address these issues going forward.

1 Introduction

Users often visit the same web pages more than once. While some of the page contents change, the vast majority of the page and associated resources (such as images) remain static. To re-download this unchanged content on every visit to the page is a waste of time and bandwidth [1]. Consequently, when a user accesses a web page, the web browser caches most content locally on the user's machine. This content can either be saved in temporary memory (RAM), which is lost as soon as the user exits the browser, or on disk, which persists even after the user exits the browser or reboots the computer. When a user visits a page repeatedly, the content is retrieved from cache instead of over the Internet. Memory caches are lost when the browser exits, so the browser uses the disk cache whenever possible.

H.-S. Lee and D.-G. Han (Eds.): ICISC 2013, LNCS 8565, pp. 496–513, 2014.
DOI: 10.1007/978-3-319-12160-4_30

Secure web sites use the HTTPS and SSL/TLS protocols to encrypt information as it travels over the Internet, to prevent an eavesdropper or man-in-the-middle from recovering or modifying the communication. Although there are no technical constraints preventing content sent over an encrypted connection from being decrypted and written to disk, it is logical to presume that if content is too sensitive to be sent over a network without encryption, then it may also be too sensitive to store unprotected on a hard drive [2].

When HTTPS was first introduced, there was no standard, unambiguous way for a web server to mark content as too sensitive to store in cache. As a result, web browser authors created their own mechanisms for a web server to restrict disk caching [2]. Some browser authors chose to, by default, never write content transferred over HTTPS to disk [3], or did not disk cache content unless a server explicitly allowed it ("opt-in") [4], while others chose to write this content to the disk cache, unless a server header explicitly prohibited it ("opt-out") [5].

We surveyed 30 sites that serve sensitive content over HTTPS, and found that 21 of those sites failed to appropriately prevent disk caching across all browsers. Of those 21, over half appear to have attempted to prevent disk caching using outdated, non-standard, or erroneous methods, while the remainder simply made no attempt. The sites surveyed included banks and other financial institutions, insurance companies, and utility companies. The sites served content that we deemed sensitive such as bank account statements, credit reports, check images, pay stubs, health and vehicle insurance information, and prescription names and dosages.

Our research found that despite the existence of reliable methods to prevent disk caching, the diversity and inconsistency across browsers in how disk caching is handled, as well as general misunderstandings within the security community, including respected sources such as OWASP, have led to wide-spread failure of web applications to reliably prevent disk caching of sensitive data. In this paper, we provide a history of inconsistent browser behavior and an understanding of that behavior as evidenced through our own verification and online references. We identify and catalog six different behaviors and techniques that effectively prevent disk caching for various versions of Internet Explorer, Firefox, Chrome, and Safari, as well as obsolete browsers such as Netscape and Mozilla (for reference purposes) – to our knowledge, no such catalog exists. We provide the best recommended actions a web developer can make to most effectively prevent disk caching across all browsers, as well as make recommendations to the security community, browser developers, and standards bodies. Lastly, we introduce a Firefox extension that end-users can install to effectively prevent disk caching, and an online resource for web application developers to test browser behaviors.

2 A Brief History

In 1997[1], the first HTTP/1.1 [6] standard was published, which standardized the header that a server must set to prevent content from being written to a disk cache. By that time, all web browser authors had already adopted either an "opt-in" HTTPS disk caching

[1] The current RFC 2616 was published in 1999, but obsoleted this older RFC 2068 which already defined Cache-control: no store.

policy, or an “opt-out” policy with multiple, non-standard ways to opt out. Despite the new standard, web developers could continue to use the old, non-standard methods and they would continue to work only in the browsers that recognized them [2].

Between the release of Netscape Navigator 3.0 in 1996, and 2008, when Google Chrome was released, the only browser with a significant market share that used an “opt-out” HTTPS disk caching policy was Internet Explorer. Internet Explorer has always been very forgiving in determining a web server’s intention that a response not be written to the disk cache. We identified four separate ways [2] that a web developer can prevent a response from being cached to disk. Only one of those ways, the header `Cache-Control: no-store`, is actually standard [6].

Encrypted web servers (HTTPS) have higher overhead and lower performance than unencrypted servers due to the need to perform encryption, and in the past this overhead was much more pronounced. For this reason, many web sites used HTTPS only when absolutely necessary, such as for sending a password or credit card information. After the sensitive transaction was completed, the sites would switch back to an unencrypted connection. Two examples are Gmail, which transmitted e-mails over unencrypted connections until 2010 [7], and Facebook, which continued to use unencrypted connections until 2012 [8]. Since HTTPS was reserved for only the most sensitive information, an “opt-in” disk caching policy was a reasonable design.

By 2011, many sites had begun using HTTPS even for non-sensitive content, and Mozilla Corporation recognized [9] that the “opt-in” HTTPS disk caching policy in Firefox was introducing a performance penalty compared to other browsers, including Google Chrome, which uses an “opt-out” policy. As a result, Firefox 4.0 and all later versions use an “opt-out” HTTPS disk caching policy [9]. A Firefox 3.6 user would be unaffected by this issue, even when browsing HTTPS sites that fail to set the necessary header, but would become affected as soon as that user updated to Firefox 4 or later.

Online banking, which is among the most security-sensitive uses of a web browser, exploded in popularity in the early 2000s. At this time, Internet Explorer had over 90 % market share, and Safari and Chrome did not exist. Internet Explorer’s only significant competitors at the time (Netscape 3.0 and later, Mozilla, and Firefox) either did not disk cache HTTPS content at all (unless a user manually modified a configuration parameter), or used an “opt-in” policy, and thus required no special treatment to prevent caching of encrypted bank pages. Many of the web sites that we tested responded with sufficient headers to prevent caching in all versions of Internet Explorer, all versions of Safari, and Firefox 3.6 and earlier, but not Firefox 4.0 and later, or any version of Chrome. We believe that ensuring that sensitive content is not cached on disk by the browser was a design goal in these web applications. While this anti-disk cache functionality worked correctly in the past, it no longer works in two of today’s most popular browsers: Chrome, and Firefox 4.0 and later. Since this has been an issue in Chrome since its release in 2008, and in Firefox since 2011, the maintainers of HTTPS sites do not appear to perform regression testing for this issue.

Today, Internet Explorer continues to follow the same HTTPS disk caching policy as it always has: enable disk caching by default, but allow four different ways to disable it. Google Chrome and Firefox, in contrast, enable disk caching by default, but allow only one way to prevent it—the one given in the standard, the header `Cache-Control: no-store`.

Google Chrome and Mozilla Firefox, together, now have over a 60 % market share on non-mobile devices [10], but many web sites still use antiquated, non-standard methods to prevent disk caching of sensitive HTTPS content that only function in Internet Explorer.

3 The Evolution of Caching Policies

Prior to HTTP/1.1 being standardized in 1997, there was no unambiguous way for a web server to instruct a client that a response should not be cached to persistent storage. Indeed, the HTTP/1.0 RFC noted [11]:

Some HTTP/1.0 applications use heuristics to describe what is or is not a "cacheable" response, but these rules are not standardized.

The cache controlling mechanisms that did exist, such as the "Expires" header, were intended to prevent a user agent from displaying stale content, and were unrelated to security. It is unnecessary to totally block the client from retaining a copy of sensitive content in memory for later reuse, instead, the objective is only to prevent the information from being written to disk.

When Netscape 1 introduced SSL and HTTPS in 1995, the browser never wrote HTTPS content to the disk cache [3]. A web server could not override this for non-sensitive content, nor could a user alter this behavior in the preferences.

This behavior changed in Netscape 2, which introduced an "opt-out" policy. Whether content was delivered over HTTP or HTTPS no longer factored into the caching decision; instead, the browser introduced a non-standard `Pragma: no-cache` response header allowing a server to prohibit the disk caching of a response. In the standard [11], `Pragma` was originally intended to be a request header, allowing a client to override any cached copies stored on intermediate proxy servers; nonetheless, introducing it as a response header at least created a way to prevent disk storage of sensitive data. However, Netscape also allowed the `Pragma: no-cache` header to be specified as a `meta http-equiv` HTML tag in the document. This was a bad design choice for two reasons: first, caching code must read the response and parse the HTML before the caching decision can be made, lowering performance; second, the tag can only be used in HTML files, and not images, JavaScript files, and so on. This "opt-out" HTTPS caching policy was incorporated by Microsoft into Internet Explorer 3 as well [2].

Possibly recognizing the potential security issue of web developers neglecting to mark sensitive data with the `Pragma` header, Netscape 3 reverted to the previous behavior of never caching HTTPS responses to disk; We verified this behavior by testing Netscape Navigator 3.04 Gold. Disk caching of HTTPS data could be re-enabled by the user in the preferences dialog, but there was still no way for a server to explicitly "opt-in" to caching of non-sensitive HTTPS content. In contrast to Netscape 3, Microsoft continued to use "opt-out" HTTPS caching in later Internet Explorer versions. Thus Netscape 3 marked the beginning of inconsistent HTTPS disk caching policies between browsers, which remains unresolved even today.

In addition to the non-standard `Pragma` header introduced by Netscape, Microsoft added support for new, standardized caching headers to Internet Explorer as they came into existence. Internet Explorer 4 added support for the `Cache-Control: no-store` header introduced in the HTTP/1.1 standard. But it also added new quirks:

- IE 4 through 9 treated the `Cache-Control: no-cache` header, intended to prevent stale responses and not a security measure, identically to the `Cache-Control: no-store` header. In version 10, `Cache-Control: no-cache` no longer prevents disk caching.
- If IE 4 through 8 made a request using HTTP/1.1, and the server responded using HTTP/1.0, any Cache-Control headers in the server's response would be ignored. This was resolved in version 9, where Cache-Control headers are recognized even when sent by an HTTP/1.0 server. Despite the fix, all Windows XP and 2003 systems contain version 8 or earlier of Internet Explorer, and are affected by this issue.
- The above HTTP/1.0 behavior is triggered by a configuration change introduced in Apache mod_ssl in 2000 (version 2.6.5) that forces a downgrade from HTTP/1.1 to HTTP/1.0 whenever the server responds to Internet Explorer over HTTPS. This configuration was intended to work around a bug in IE 5's handling of HTTP/1.1 keep-alive connections. In 2010, long after the Internet Explorer bug was patched in version 6, Apache finally updated the workaround to exclude unaffected releases [12]. However, this configuration change has not yet percolated to all Linux distributions' standard branches of Apache, including the latest version of CentOS[2] as of this writing, 6.4.

Netscape continued with the policy of not disk caching HTTPS content by default throughout versions 3 and 4 of their browser, the last release of which occurred in 2002. Despite this, Netscape retained vestigial support for the `Pragma: no-cache` header introduced in version 2—in case the user modified the preferences to enable persistent HTTPS caching. This support was dropped when the Mozilla project began a browser rewrite in 1998, but with little consequence at the time, since the rewritten browser never cached HTTPS content by default [13].[3]

After these changes in the mid-1990s, browser caching policies, while still inconsistent and only partially following standards, did stabilize. Apple released Safari in 2003, which to this day never writes HTTPS content to the disk cache. The iOS version also follows this policy.

The stability came to an end in 2008, when Google released the Chrome browser. Chrome, and its mobile variant, Android Browser, have the most aggressive HTTPS disk caching policy ever created at the time. Content is always written to the disk cache unless one of two conditions are met: (1) the response includes the header `Cache-Control: no-store`, or (2) the server has an invalid certificate [14]. No support is included for non-standard headers supported by Internet Explorer (i.e., `Pragma:`

[2] http://mirror.umd.edu/centos/6.4/updates/i386/Packages/mod_ssl-2.2.15-28.el6.centos.i686.rpm

[3] This page [13] shows that `browser.cache.disk_cache_ssl` was set to false in revision 1.1 when Netscape first released source.

`no-cache` and `Cache-Control: no-cache`), which at the time of Chrome's release, was the only other web browser that even cached HTTPS content at all by default.

Concurrent with the release of Chrome, Mozilla began loosening the Firefox HTTPS disk caching policy as well. In Firefox 3, Mozilla introduced a unique caching policy, that in our opinion represented the best trade-off between security and performance. HTTPS continued to be treated as an indicator that content should not be disk cached, but Firefox now allowed servers to explicitly "opt-in" to caching, by including the header `Cache-Control: public` [3]. While originally intended for multiuser caching proxies, `Cache-Control: public` is defined as:

Indicates that the response is cacheable by any cache, even if it would normally be non-cacheable or cacheable only within a non-shared cache.

Thus the presence of this header is a good indicator that content is non-sensitive and safe to cache.

Still, Mozilla modified the HTTPS caching policy once again in Firefox 4, this time to cache all HTTPS content unless it is explicitly labeled as sensitive using `Cache-Control: no-store`, effectively reversing the behavior of Firefox 3. Paradoxically, this meant that the original functionality of the `Pragma: no-cache` header introduced by Firefox's ancestral Netscape browser was now only supported by Internet Explorer.

All of the different ways we found to control disk caching of HTTPS content are shown in Table 1.

Table 1. Variants of headers or HTML meta tags used to enable or prevent disk caching of HTTPS content, and listings of browsers that support each one.

Header or tag	Supporting browsers
None needed—No HTTPS disk caching by default	Netscape 1, 3 + , Firefox 1-3.5, Safari
Pragma: no-cache header (opt-out)	Netscape 2, IE 3+
Pragma: no-cache meta tag (opt-out)	Netscape 2, IE 3+
Cache-Control: no-cache header (opt-out)	IE 4-9
Cache-Control: no-store header (opt-out)	IE 4 + , Firefox 4 + , Chrome 1+
Cache-Control: public header (opt-in)	Firefox 3-3.5

4 Current Caching Policies by Browser

Disk caching of HTTPS-delivered web pages varies by web browser. Here, we discuss the policies of four browsers that we tested.

Internet Explorer. Microsoft Internet Explorer caches HTTPS-delivered content to disk, unless one or more of the following are present [2]:

- The HTTP header `Cache-Control: no-store`.
- In version 9 and earlier only, The HTTP header `Cache-Control: no-cache`.
- The HTTP header `Pragma: no-cache`.

- The HTML tag `<META HTTP-EQUIV="Pragma" CONTENT="no-cache">`. Microsoft discourages the use of this method; it may not work properly for pages larger than 32 kb [13].

Note that the Cache-Control header cannot be set using an HTML `<META HTTP-EQUIV>` tag. Additionally, Internet Explorer interprets some of these headers differently, depending upon whether the page was delivered using HTTPS or HTTP [2].

We verified that using the 32-bit version of Internet Explorer 10.0.9200.16635 on 64-bit Windows 7, HTTPS content is disk cached unless the server sends the `Pragma: no-cache` header or `Cache-Control: no-store` header, or the document contains the `Pragma: no-cache` header in an HTML `meta http-equiv` tag. We verified that using the 32-bit version of Internet Explorer 9.0.8112.16421, HTTPS content is disk cached unless the server sends the `Cache-Control: no-cache` header, or the response employs either of the two methods described for IE 10.

Firefox. Prior to version 4.0, Mozilla Firefox (and its predecessors, including Mozilla and Netscape) either never cache HTTPS pages to disk at all [3] or cache only pages sent with:

- The HTTP header `Cache-Control: public`.

Firefox contains a hidden browser preference, `browser.cache.disk_cache_ssl`, that when set to `true`, switches Firefox from the previous, cautious policy above, to a new policy that strictly follows the HTTP standard, disk caching all content unless specifically instructed not to do so by the server. In 2011, the default value of this preference was switched from `false` to `true` [9]. As a result, Firefox 4.0 and all later versions cache HTTPS-delivered content to disk, unless the following is present:

- The HTTP header `Cache-Control: no-store`.

We verified that using the 32-bit version of Mozilla Firefox 3.6.28 on 64-bit Windows 7 (and earlier), HTTPS content is not disk cached unless the server sends the `Cache-Control: public` header. We verified that using the 32-bit version of Mozilla Firefox 21.0 on 64-bit Windows 7, Mozilla Firefox 21.0 on Mac OS X 10.7.5, and Mozilla Firefox 21.0 on Android 2.3.6, HTTPS content is disk cached unless the server sends the `Cache-Control: no-store` header.

Chrome. Google Chrome caches HTTPS-delivered content to disk, unless the following is present:

- The HTTP header `Cache-Control: no-store`.

We verified that when using Google Chrome 27.0.1453.94 m on Windows 7, or the Browser app in Android 2.3.6 (which is based on Chrome), HTTPS content is disk cached unless the server sends the `Cache-Control: no-store` header.

Safari. Apple Safari does not cache HTTPS-delivered content to disk, regardless of any headers sent by the server. We tested the mobile version of Safari on an iPad 2, and the HTTPS caching behavior was identical to the desktop version.

We verified that using Safari 6.0 (7536.25) on Mac OS X 10.7.5, and Mobile Safari on iOS 5.1.1, HTTPS content is never disk cached.

A word about private browsing modes. Virtually all web browsers now include a "private browsing" mode, that in addition to preventing browsing history from being retained, disables the disk cache entirely. While sufficient for a user to avoid this issue, we do not consider advising users to use private browsing to be a reasonable solution for several reasons. First, private browsing modes are not the default, and must manually be enabled by a user. Second, other aspects of private browsing, such as not retaining persistent cookies, break useful functionality in web sites, such as remembering usernames or remembering the computer to avoid answering security questions on each login. Third, since private browsing disables the disk cache entirely, it has negative side effects on the performance of the Internet as a whole, since even unencrypted HTTP content must be re-downloaded if the browser has been closed.

5 Reliably Preventing Disk Caching

Due to the historical inconsistency and confusion surrounding HTTPS and disk caching, it is worth briefly mentioning how to most reliably prevent disk caching of an HTTPS response. To do so, the web server should be configured to send the following:

- The response header `Pragma: no-cache`.
- The response header `Cache-Control: no-store`.

The Pragma header covers the special case of HTTP/1.0 servers and Internet Explorer 8. The Cache-Control header, as specified in the HTTP standard, covers all other cases, including standards-compliant browsers that may begin caching HTTPS content in the future (e.g., Safari). As both older Apache servers and IE 8 browsers are decommissioned over time, the Pragma header will no longer be needed.

6 Site Survey

Methodology. We tested thirty secure, password-protected sites that displayed sensitive personal information in a web browser. This involved accessing SSL-protected websites as an authorized user, logging out of the site, and closing the browser. Then, we reopened the browser, placed it in offline mode, and checked the disk cache for entries containing sensitive data.

Initial Results. As of April 25, 2013, twenty-one of the thirty sites tested were not sending the `Cache-Control: no-store` header required by the HTTPS standard to prevent disk caching of sensitive data. Some were not sending any caching-related headers at all, while others were sending caching headers that prevent disk caching only in Internet Explorer, or other headers not relevant to web browser caches.

The sites shown in Table 2 sent sensitive information with both of the headers `Cache-Control: no-cache`, and `Pragma: no-cache`, which together, prevent disk caching in Internet Explorer, but not Firefox or Chrome.

Table 2. Sites sending sensitive data with the headers Pragma: no-cache and Cache-Control: no-cache.

Site	Sensitive data
ADP	Partial SSN, name, address, financial data
BGE	Name, address, account number, account balance
M&T Bank Wealthcare	Name, account number, account balance
Scottrade	Account number, account balance
TreasuryDirect	Partial SSN, name, address, phone number
Verizon Wireless	Call details

The sites shown in Table 3 sent sensitive information with the header `Cache-Control: no-cache` which prevents disk caching in Internet Explorer 9 and earlier, but not Internet Explorer 10, Firefox or Chrome.

Table 3. Sites sending sensitive data with the header Cache-Control: no-cache.

Site	Sensitive data
BB&T	Name, partial account numbers, account balances
Liberty Mutual	Name, policy number, policy limits, account balances
PayPal	Name, address, phone number

The sites shown in Table 4 sent sensitive information with the header `Cache-Control: private`, which has no effect on whether or not a web browser caches the information to disk.

Table 4. Sites sending sensitive data with the header Cache-Control: private.

Site	Sensitive data
Allstate	Auto insurance policies
eBillity	Worker summary reports
eRenterPlan	Name, address, phone number

Lastly, the sites shown in Table 5 sent sensitive information without any cache-related HTTP headers at all.

Figures 1, 2, 3, 4, 5, and 6 in Appendix A show screenshots of some of the sensitive data we recovered from the disk cache.

Table 5. Sites sending sensitive information without cache-related headers.

Site	Sensitive data
Argus Health	Prescription claims
Boscov's Charge Card	Statements, full account numbers
Equifax	Full credit reports
GEICO	Partial SSN, DOB, name, address
MetLife	Name, policy number, policy amount, beneficiaries
PNC Bank	Check images
T. Rowe Price	401(k) balances
Toyota Financial	Name, address, account number, VIN
Trade King	Account number, balance

7 Updates

We notified each company in April, 2013, by email to the security- or phishing-related email address, or when email was not available, using a web-based contact form. The following companies acknowledged our advisories with a non-automated response:

- Argus Health.
- M & T Bank.
- PayPal.

Only BB&T has made any identifiable progress in over four months since notification toward implementing proper cache control behavior. The account summary page is now sending `Cache-Control: no-store`, but check images are still sent with inadequate protections.

8 Observations and Concerns

We believe that the amount of personal data that is currently being written to the disk cache when visiting these sites is alarming. It is important to note the distinction between a user consciously selecting a "save to disk" option, e.g., to save a bank statement, and content silently being written to the disk cache without users' knowledge. Non-technical users likely believe that if, after visiting a site and viewing personal data, they logout and close their browsers, that their data will be purged. Our findings prove this assumption incorrect in 70 % of the cases tested.

Based on the quantity of sites (twelve of twenty-one) that sent at least one cache-related header, even if it was not the one mandated by the standard to prevent disk caching, we do not believe that it is intended by these industries that this content be written to the disk cache. More significantly, the maintainers of these sites may erroneously believe that they have set the required headers to prevent disk caching, based on outdated and incorrect information published on the Internet. One tutorial [14] correctly states the purpose of all of these headers, but does not put them in the proper context with regard to HTTPS, stating "SSL pages are not cached (or decrypted) by proxy caches," which, while true for proxies, does not address the behavior of browsers. An OWASP page [15] incorrectly asserts that "If a web page is delivered over SSL, no content can be cached." When even the security community makes outdated and incorrect assumptions about this issue, it is unrealistic that more generally-focused web developers will do better.

Web browser authors, with the most striking example being Mozilla, seem to dismiss the current reality of servers sending sensitive information without the header needed to prevent it from being cached to disk. A comment on the bug report involving the change to Firefox 4's SSL caching policy by a member of Mozilla Corporation's security team stated [9]:

Among sites that don't use cache-control:no-store, the correlation between "SSL" and "sensitive" is very low.

Our findings show that this assertion does not hold when real-world sites are examined, even two years after the change.

The fact that the unencrypted, disk cached data is only stored on the user's personal machine should not be discounted. The possibilities for this information to be exposed are numerous: malware infections, theft of laptops and mobile devices, theft of physical backup media or compromise of "cloud" backup services, shared machines and user accounts [17], and of course, shared computers in libraries, hotels, and Internet cafes. An Intel-sponsored Ponemon Institute study estimated the cost of recovering from the loss or theft of a single laptop as $49,246 [16], and a Lookout Mobile Security study estimated that lost and stolen phones cost consumers more than $30 billion in 2012 [17].

9 Recommendations

To Web Developers and Web Framework Authors. Developers of web applications and web frameworks should audit all existing code to ensure that sensitive data is labeled with the appropriate caching directives. Professionals in these fields must become more familiar with the fine details of the HTTP standard, and assume that browser software will always make performance vs. security trade-offs against security. Proper security assessments of sites containing sensitive information should be conducted regularly, and an examination of disk cached content across all supported browsers should be part of that assessment process.

To the Security Community. All existing guidance and advice in regard to the HTTPS caching issue should be revised to reflect the reality of the HTTP standard. Security professionals should be cautious in making assertions or recommendations based on working knowledge alone, and be sure to consult the relevant standards and perform testing to back up their beliefs.

To Web Browser Authors. In a time where security threats and identity theft are rampant, all browsers should adopt an "opt-in" only policy for caching sensitive data to disk; and further, users should have an easily accessible option to refuse any or all "opt-in" directives. At the very least, we recommend that browsers with a very strict "opt-out" HTTPS disk caching policy, such as Firefox and Chrome, consider interpreting the

Pragma HTTP header and meta tag supported by Internet Explorer, as well. Internet Explorer has been disk caching HTTPS content for far longer than either of these browsers, so many sites seem to have been developed with IE-centric security assumptions in mind.

To Standards Committees. We recommend that standards bodies incorporate sound security principles, such as secure-by-default, defense-in-depth, and fail safe, into future standards. Traditionally, standards authors have attempted to maintain the layered architecture of Internet standards, and avoid tightly coupling an application-layer protocol like HTTP to the layers below. Indeed, RFC 2616 [put back ref?], the latest version of HTTP/1.1, mentions “SSL” once and does not mention “HTTPS” at all. While avoiding any consideration of whether an encrypted or unencrypted connection is used might make for a cleaner design with fewer special cases, it has practical security consequences. If HTTP/1.1 had simply specified that persistent caching was disabled by default on encrypted connections, and specified a header allowing a server to mark content as non-sensitive, then this entire issue could have been avoided.

To End-Users. Users should make the following configuration changes, depending on each browser, keeping in mind there may be performance trade-offs associated with these actions:

Internet Explorer. Internet Explorer already abides by most web application attempts to prevent disk caching. To further restrict what can be cached, a user can open Internet Options, choose the “Advanced” tab, and under “Security,” check “Do not save encrypted pages to disk.” This option may have unwanted side effects, such as interfering with file downloads from HTTPS sites. Alternatively, use “InPrivate Browsing” mode.

Firefox. Install our “HTTPS Caching Controller” Firefox add-on,[4] which adds a toolbar button allowing disk caching of SSL content to be disabled or enabled at any time. This add-on works only on the desktop version of Firefox. Manually, or on the mobile version, navigate to `about:config`, locate the preference `browser.cache.disk_cache_ssl`, and set the value to `false`. Alternatively, use “Private Browsing” mode.

Chrome. Google Chrome does not appear to have configurable functionality to limit the disk caching of HTTPS content (without affecting HTTP content) without modifying the source code. A workaround is to use “Incognito” mode, which prevents *all* disk caching.

The mobile Android Browser is similar. Android users can switch to another browser, such as the mobile version of Firefox, or use “Incognito” mode.

Safari. Safari users (both desktop and mobile) need not take any action, since, as of this writing, Safari does not cache any content transferred over HTTPS.

General. In addition to taking these precautions, never log into account-related or other security-sensitive sites from a computer or other device you do not own and control.

[4] http://securityevaluators.com/content/case-studies/caching/extension.jsp

10 Conclusions and Future Work

We have shown here, through direct verification and through online investigation, that the history of web browser caching behavior is a complicated one. The inconsistency across browser platforms and even across individual browser versions, has caused security and development communities much confusion, as evidenced by online sources and the alarming results of our study: that over 70 % of HTTPS-protected sites containing highly sensitive data fail to properly prohibit disk caching, and of them over 50 % appear to desire such prohibition.

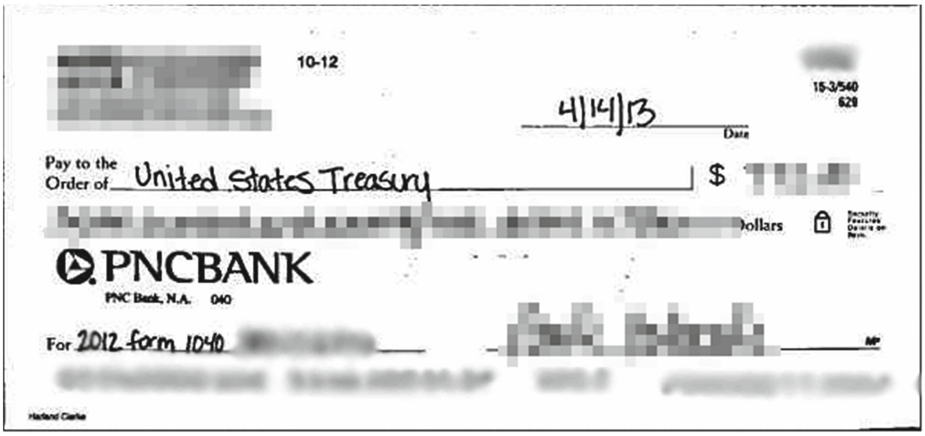
We have identified the actual disk caching behavior of the four most popular web browsers, and suggest to web developers the most effective ways to prevent disk caching of sensitive content across all browsers.

For end-users, we have provided a Firefox extension that effectively prohibits disk caching of user-chosen sensitive data, rather than relying on the web application itself to make the appropriate decisions.

Moving forward, standards bodies should consider updating the HTTP standard so that the persistent caching of HTTPS data follows an "opt-in" policy, that is, the standard should recommend never caching HTTPS-protected content unless the web application specifically indicates that data is safe to cache.

Our data set consisted of 30 web sites, and additional statistical study could be performed to determine how many sites fail to properly prohibit the disk caching of sensitive data. Furthermore, given the lack of response to our disclosure of this information, it would be interesting to statistically gauge the response time of these organizations.

Appendix A


[redacted] 10-12 [redacted]

15-3/540
628

4/14/13
Date

Pay to the Order of United States Treasury $ [redacted]

[redacted] Dollars Security Features Details on Back.

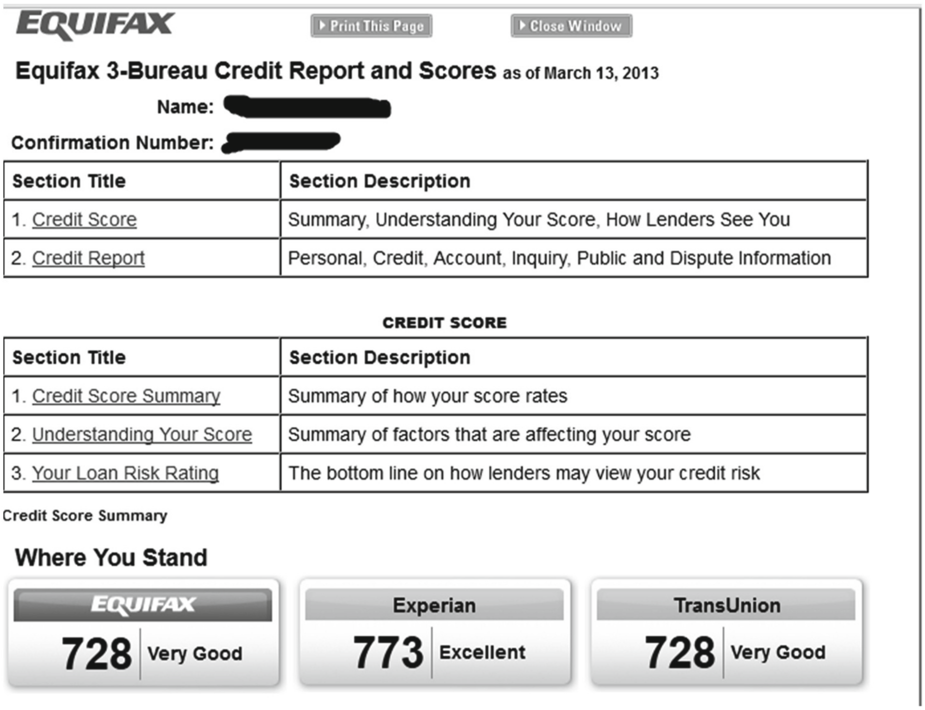
PNCBANK
PNC Bank, N.A. 040

For 2012 form 1040 [redacted] [redacted] MP

[redacted]

Harland Clarke

Fig. 1. Check image from PNC.



Equifax 3-Bureau Credit Report and Scores as of March 13, 2013

Name: [redacted]

Confirmation Number: [redacted]

Section Title	Section Description
1. Credit Score	Summary, Understanding Your Score, How Lenders See You
2. Credit Report	Personal, Credit, Account, Inquiry, Public and Dispute Information

CREDIT SCORE

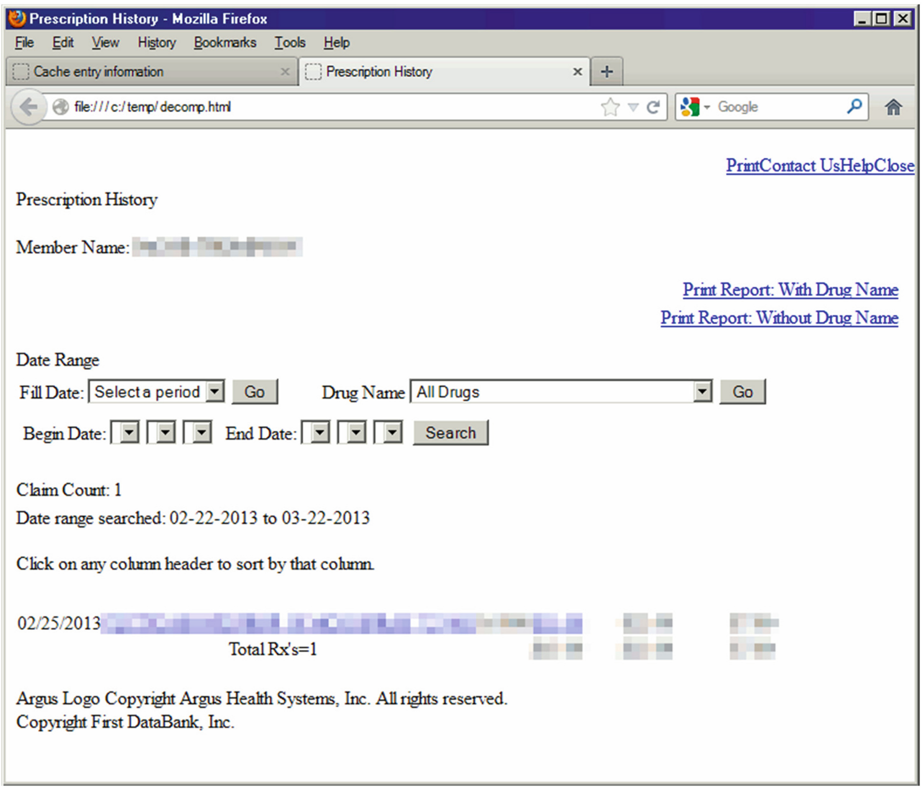
Section Title	Section Description
1. Credit Score Summary	Summary of how your score rates
2. Understanding Your Score	Summary of factors that are affecting your score
3. Your Loan Risk Rating	The bottom line on how lenders may view your credit risk

Credit Score Summary

Where You Stand

EQUIFAX	Experian	TransUnion
728 Very Good	773 Excellent	728 Very Good

Fig. 2. Full credit report from Equifax.


Prescription History - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Cache entry information
Prescription History
file:///c:/temp/decomp.html
Google

PrintContact UsHelpClose

Prescription History

Member Name: [redacted]

Print Report: With Drug Name
Print Report: Without Drug Name

Date Range
Fill Date: Select a period Go Drug Name All Drugs Go
Begin Date: End Date: Search

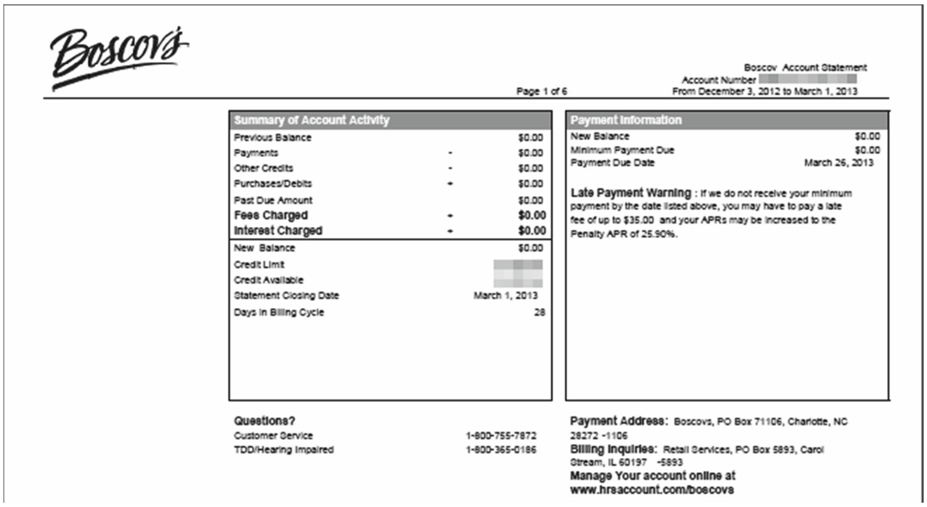
Claim Count: 1
Date range searched: 02-22-2013 to 03-22-2013

Click on any column header to sort by that column.

02/25/2013 [redacted]
Total Rx's=1 [redacted]

Argus Logo Copyright Argus Health Systems, Inc. All rights reserved.
Copyright First DataBank, Inc.

Fig. 3. Prescription information from Argus



Boscov Account Statement
Account Number [redacted]
Page 1 of 6 From December 3, 2012 to March 1, 2013

Summary of Account Activity		
Previous Balance		$0.00
Payments	-	$0.00
Other Credits	-	$0.00
Purchases/Debits	+	$0.00
Past Due Amount		$0.00
Fees Charged	+	**$0.00**
Interest Charged	+	**$0.00**
New Balance		$0.00
Credit Limit		[redacted]
Credit Available		[redacted]
Statement Closing Date		March 1, 2013
Days in Billing Cycle		28

Payment Information	
New Balance	$0.00
Minimum Payment Due	$0.00
Payment Due Date	March 26, 2013

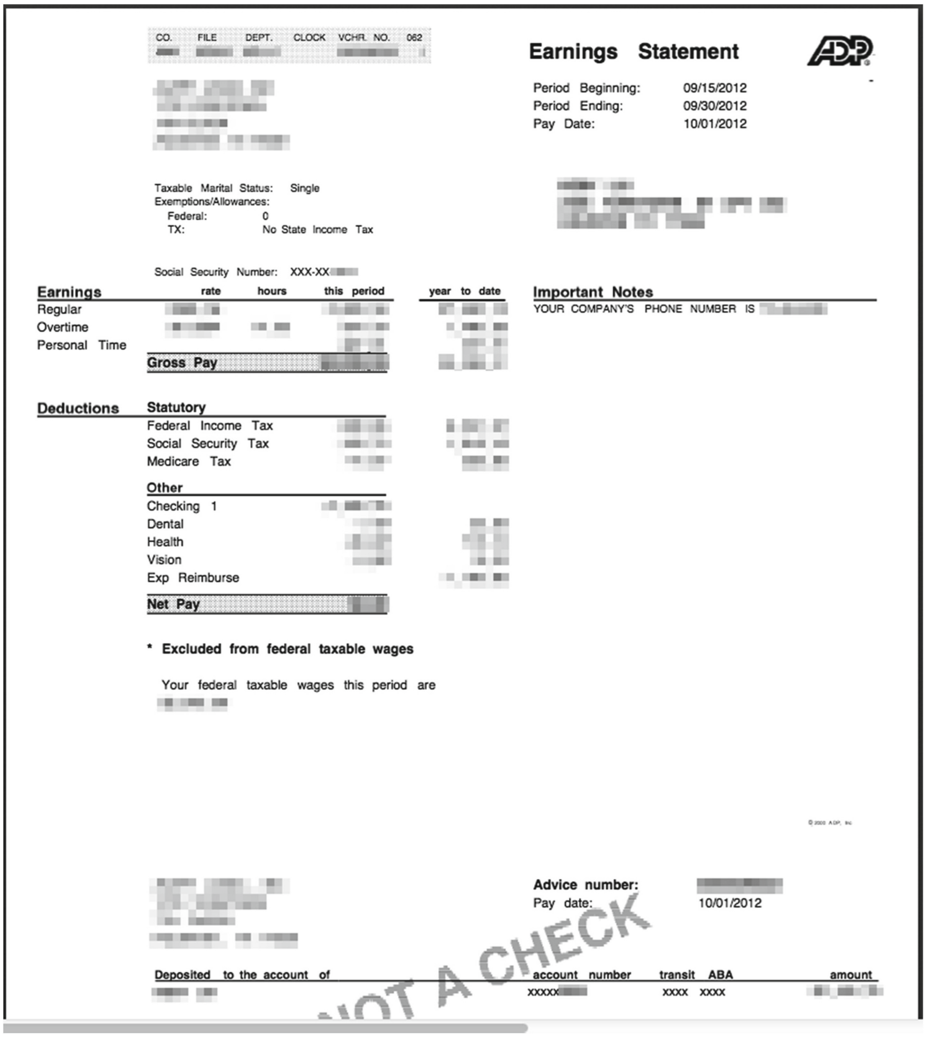
Late Payment Warning : If we do not receive your minimum payment by the date listed above, you may have to pay a late fee of up to $35.00 and your APRs may be increased to the Penalty APR of 25.90%.

Questions?

Customer Service	1-800-755-7872
TDD/Hearing Impaired	1-800-365-0186

Payment Address: Boscovs, PO Box 71106, Charlotte, NC 28272 -1106
Billing Inquiries: Retail Services, PO Box 5893, Carol Stream, IL 60197 -5893
Manage Your account online at www.hrsaccount.com/boscovs

Fig. 4. Credit card account statement from Boscov's



CO.	FILE	DEPT.	CLOCK	VCHR. NO.	062
[redacted]	[redacted]	[redacted]		[redacted]	

Earnings Statement

Period Beginning:	09/15/2012
Period Ending:	09/30/2012
Pay Date:	10/01/2012

[redacted]

[redacted]

Taxable Marital Status: Single
Exemptions/Allowances:
Federal: 0
TX: No State Income Tax

Social Security Number: XXX-XX[redacted]

Earnings	rate	hours	this period	year to date
Regular	[redacted]		[redacted]	[redacted]
Overtime	[redacted]	[redacted]	[redacted]	[redacted]
Personal Time			[redacted]	[redacted]
Gross Pay			[redacted]	[redacted]

Deductions	Statutory	this period	year to date
	Federal Income Tax	[redacted]	[redacted]
	Social Security Tax	[redacted]	[redacted]
	Medicare Tax	[redacted]	[redacted]
	Other		
	Checking 1	[redacted]	
	Dental	[redacted]	[redacted]
	Health	[redacted]	[redacted]
	Vision	[redacted]	[redacted]
	Exp Reimburse		[redacted]
	Net Pay	[redacted]	

Important Notes
YOUR COMPANY'S PHONE NUMBER IS [redacted]

* **Excluded from federal taxable wages**

Your federal taxable wages this period are [redacted]

© 2000 ADP, Inc

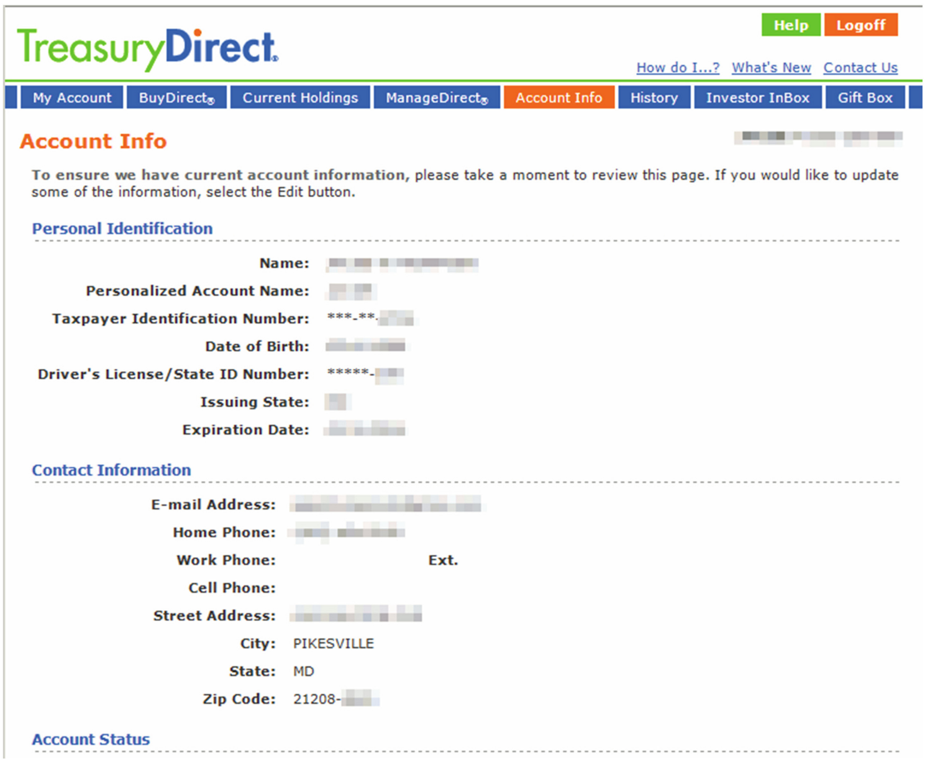
[redacted]

Advice number: [redacted]
Pay date: 10/01/2012

Deposited to the account of	account number	transit ABA	amount
[redacted]	XXXXX[redacted]	XXXX XXXX	[redacted]

NOT A CHECK

Fig. 5. Paystub from ADP.



Account Info [redacted]

To ensure we have current account information, please take a moment to review this page. If you would like to update some of the information, select the Edit button.

Personal Identification

Name:	[redacted]
Personalized Account Name:	[redacted]
Taxpayer Identification Number:	***-**-[redacted]
Date of Birth:	[redacted]
Driver's License/State ID Number:	*****-[redacted]
Issuing State:	[redacted]
Expiration Date:	[redacted]

Contact Information

E-mail Address:	[redacted]
Home Phone:	[redacted]
Work Phone:	Ext.
Cell Phone:	
Street Address:	[redacted]
City:	PIKESVILLE
State:	MD
Zip Code:	21208-[redacted]

Account Status

Fig. 6. Account information from Treasury Direct.

References

1. Barish, G., Obraczke, K.: World Wide Web caching: trends and techniques. Commun. Mag. **38**(5), 178–184 (2000)
2. Microsoft: How to prevent caching in Internet Explorer, Microsoft. http://support.microsoft.com/kb/234067. Accessed 26 July 2013
3. Appel, S.: Secure sockets layer discussion list FAQ v1.1.1, faqs.org, 16 November 1998. http://www.faqs.org/faqs/computer-security/ssl-talk-faq/. Accessed 26 July 2013
4. Mozilla: Firefox ignores “Cache-control: public” header on TLS connections, Mozilla, 19 July 2006. https://bugzilla.mozilla.org/show_bug.cgi?id=345181. Accessed 26 July 2013
5. Microsoft: Cannot open files on secure servers, Microsoft. http://support.microsoft.com/kb/254324. Accessed 26 July 2013
6. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1 (RFC 2068), IETF (1997)
7. Schillace, S.: Default https access for Gmail, Google, 12 January 2010. http://gmailblog.blogspot.com/2010/01/default-https-access-for-gmail.html. Accessed 25 July 2013
8. Rice, A.: Keeping users safe, Facebook, 13 May 2011. https://developers.facebook.com/blog/post/499/. Accessed 26 July 2013

9. Mozilla: Should cache SSL content to disk even without Cache-Conrol: public, Mozilla, 30 November 2009. https://bugzilla.mozilla.org/show_bug.cgi?id=531801. Accessed 26 July 2013
10. Everyone: Usage share of web browsers, Wikipedia. http://en.wikipedia.org/wiki/Browser_market_share. Accessed 25 July 2013
11. Berners-Lee, T., Fielding, R., Frystyk, H.: Hypertext transfer protocol - HTTP/1.0 (RFC 1945), IETF (1996)
12. The Apache Software Foundation: Revision 966055, The Apache Software Foundation, 20 July 2010. http://svn.apache.org/viewvc?view=revision&revision=966055. Accessed 26 July 2013
13. Microsoft: "Pragma: No-cache" tag may not prevent page from being cached, Microsoft. http://support.microsoft.com/kb/222064. Accessed 26 July 2013
14. Nottingham, M.: Caching tutorial for web authors and webmasters, 06 May 2013. http://www.mnot.net/cache_docs. Accessed 26 July 2013
15. OWASP: OWASP Application Security FAQ, OWASP, 22 April 2007. https://www.owasp.org/index.php/OWASP_Application_Security_FAQ#Am_I_totally_safe_with_these_directives.3F. Accessed 26 July 2013
16. Ponemon Institute: The billion dollar lost laptop problem, Ponemon Institute, (2010)
17. Lookout: Lookout projects lost and stolen phones could cost U.S. consumers over $30 billion in 2012, 21 March 2012
18. Chromium: Contents of /releases/1.0.154.53/src/net/http/http_cache.cc, Chromium, 26 July 2008. http://src.chromium.org/viewvc/chrome/releases/1.0.154.53/src/net/http/http_cache.cc?revision=14. Accessed 26 July 2013