Cache Cookies for Browser Authentication (Extended Abstract)

Ari Juels
RSA Laboratories and
RavenWhite Inc.
ajuels@rsasecurity.com

Markus Jakobsson
Indiana University and
RavenWhite Inc.
markus@indiana.edu

Tom N. Jagatic
Indiana University
tjagatic@iu.edu

Abstract

Like conventional cookies, cache cookies *are data objects that servers store in Web browsers. Cache cookies, however, are unintentional byproducts of protocol design for browser caches. They do not enjoy any explicit interface support or security policies.*

In this paper, we show that despite limitations, cache cookies can play a useful role in the identification and authentication of users. Many users today block conventional cookies in their browsers as a privacy measure. The cache-cookie tools we propose can help restore lost usability and convenience to such users while maintaining good privacy. As we show, our techniques can also help combat online security threats such as phishing and pharming that ordinary cookies cannot. The ideas we introduce for cache-cookie management can strengthen ordinary cookies as well.

The full version of this paper may be referenced at www.ravenwhite.com.

Keywords: cache cookies, personalization, malware, pharming, phishing, privacy, Web browser

1 Introduction

A conventional cookie is a piece of data stored in a specially designated cache in a Web browser. Cookies can include user-specific identifiers or personal information (e.g., this user is over 18 years of age). Servers typically employ cookies to personalize Web pages. For example, when Alice visits the Web site X, the domain server for X might place a cookie in Alice's browser that contains the identifier "Alice." When Alice visits X again, her browser releases this cookie, enabling the server to identify her automatically.

A cache cookie, by contrast, is not an explicit browser feature. It is a form of persistent state in a browser that a server can access in unintended ways. There are many different forms of cache cookies; they are byproducts of the way that browsers maintain various caches and access their contents.

For example, one type of cache cookie, which we believe to be new to the literature and focus on here, is based on Temporary Internet Files (TIFs). TIFs are data objects – such as images – cached locally in standard browsers. Their function is to accelerate browsing speeds: If the browser is to display a data object present as a TIF, it can access the object locally, rather than pulling it from a server. TIFs can be turned into cache cookies, i.e., persistent, server-accessible data objects. By caching a particular TIF X associated with its domain, a server effectively writes a bit into the browser of a particular user. By causing a client to display a Web page containing X, and then seeing whether the client requests X, the server can determine if X is present as a TIF in the client's browser. Thus by testing for the presence or absence of TIF X, a server can read a bit from the browser.

A cache cookie can function very much like an ordinary cookie. It is common for servers to plant cookies containing secret values in the browsers of users. These cookies help a server authenticate a user – or, more precisely, her browser. Cache cookies, as we show, can serve much the same goal.

1.1 Our work: Cache cookies as authenticators

Cookies were designed not for authentication, but as a convenient way to pass state. They have been co-opted in many systems to achieve security goals, chiefly as *authenticators* to supplement passwords. We take the same approach to cache cookies: We co-opt them for the unintended benefits of user identification and authentication.

Because cookies (and similar sharable objects) are fully accessible by the domain that set them, they are vulnerable to *pharming*. A pharming attack creates an envi-

Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)
1081-6011/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY

ronment in which a browser directed to the Web server legitimately associated with a particular domain instead connects to a spoofed site. A pharmer can then harvest the browser-cached objects associated with the attacked domain. Even SSL offers only modest protection against such cookie harvesting. A pharmer can use an incorrect certificate and simply rely on users' tendency to disregard browser warnings.[1] But a pharmer can use an incorrect certificate and simply rely on the tendency of users to disregard browser warnings.

We show how to use cache cookies in ways that do not rely on server domains and can therefore resist many pharming attacks. The basis for our work is a new conceptual framework in which cache cookies underpin a general, virtual memory structure within a browser. We refer to this type of structure as *cache-cookie memory*, abbreviated *CC-memory*.

A key feature of CC-memory is that it spans a huge space. It is a virtually addressed memory structure, not a physically addressed one. Thus its size is exponential in the bit-length of browser resource names like URLs. So large is the space of CC-memory in a browser that a server can only access a negligible portion, and *an attacker cannot feasibly read more than a negligible portion of CC-memory.* We propose new techniques for privacy-enhanced identifiers and user-authentication protocols that resist brute-force attacks against browser caches. Importantly, our techniques require no special-purpose client-side software.

1.2 Related work

While we emphasize the use of cache cookies for authentication in this paper, most of the literature thus far has treated cache cookies purely in light of their threat to privacy.

Felten and Schneider first brought the problems of invasive cache cookies to light [2], and indeed first coined the term "cache cookies." They showed how a server can detect the presence of a given image file in a browser cache, and thus use cached images as cache cookies. Their techniques are based on timing analysis, however, and somewhat difficult to implement.

Clover [1], however, brought to light more easily manipulated cache cookies based on browser histories. A side-effect in Cascading Style Sheets (CSS) (a framework for presenting Web content) permits a server to embed code in a Web page that determines whether or not a browser contains a particular URL in its history. For example, *any* server can determine if Alice has visited the specific Web page www.arbitrarysite.com/randompath/index.html. Additionally, a server can effectively write a URL X into the browser history of a client by directing the client to the URL X (in, e.g., an invisible frame). Thus, browser-history entries can function as cache cookies.

A related, common form of tracking used by marketers today is what is known as a "Web bug," a client-specific HTML link to an (invisible) image in e-mail. By downloading the image, a client alerts a server to the opening of the e-mail. Web bugs can also be planted in Web pages.

More recently, Jackson et al. [3] examine the privacy impact of cache cookies and related browser features, and present a unified view of cross-domain tracking threats to users. They also identify new facets to cache cookies, such as *entity tags* (Etags), which we discuss below. Jackson et al. propose browser extensions to enforce consistent privacy policies across a range of cross-domain tracking methods.

Again, our emphasis in this paper is on the *positive* face of cache cookies. We propose ways to use cache cookies beneficially without exacerbating existing privacy problems.

Organization: In section 2, we present our framework for CC-memory, along with some new implementation options. We introduce schemes for user identification and authentication in section 3. We present supporting experiments in section 4, and conclude in section 5.

2 Cache-Cookie Memory Management

We now explain how to construct CC-memory structures. As explained above, CC-memory is a general read/write memory structure in a user's browser. We use cache cookies based on TIFs as an illustrative example, but the same principles apply straightforwardly to other types of cache cookies.

A server can, of course, plant any of a wide variety of TIFs by giving them appropriate URLs. For example, a server operating the domain www.arbitrarysite.com can plant in a browser a GIF with the URL "www.arbitrarysite.com/Z.gif," where Z can be any URL-compliant string. Thus, a server can create a CC-memory structure over the space of URLs of the form, e.g., "www.arbitrarysite.com/Z.gif", where $Z \in \{0,1\}^{l+1}$. In other words, Z is an index into the CC-memory space. In practice, this virtual-memory space can be enormous – larger than a cryptographic key space. (Current versions of IE, for instance, support 2048-bit URL paths.)

[1] It is generally difficult for a pharmer to obtain the private key corresponding to a legitimate SSL certificate for a domain under attack. Pharmers have been known to obtain fraudulent certificates, however [4]. And in principle a pharmer dupe a user to installing an invalid certificate in her browser.

When l is sufficiently large – in practice when cache cookies are 80 or so bits long – CC-memory is large enough to render brute-force search by browser sniffing impractical. Suppose, for example, that a server plants a secret, k-bit string $x = x_0x_1 \ldots x_k$ into a random location in CC-memory of the browser of a given user. It is infeasible for a second server interacting with the user to learn x — or even to detect its presence. A server can thus hide cache cookies from adversaries. As we explain in the full paper, CC-memory can assume any of a variety of virtual memory structures, and can support not just reading and writing, but also erasure and re-writing.

TIF-based cache cookies: As explained above, *Temporary Internet files* (TIFs) are files containing objects, e.g., images embedded in Web pages. Browsers cache these files to support faster display when a user revisits a Web page. TIFs have no associated expiration, but browsers cap the disk space devoted to TIFs and delete them to maintain this cap. Thus TIF persistence varies among users.

To place a TIF X in a browser cache, a server can serve content that causes downloading of X. It can verify whether or not a browser contains X in its cache by displaying a page containing X. If X is not present in its cache, then the browser will request it; otherwise, the browser will not pull X, but instead retrieve its local copy. In order not to change the state of a cache cookie for whose presence it is testing, a server must in the former case withhold X. This triggers a "401" error, but manipulation of TIFs can occur in hidden windows, unperceived by users.

Cache cookies based on TIFs restrict read privileges, a useful privacy feature. When a browser requests a TIF X, it sends a request to the domain associated with X, *not* to the server displaying content containing X. Thus TIF-based cache cookies are like first-party cookies: Only the site in control of the domain for X can detect the presence of X in a browser cache. (Cross-domain timing attacks [2] can undermine the first-party property for TIFs, but are challenging to mount.)

A notable limitation of TIFs is that they cannot be manipulated over SSL. As a security measure, HTTPS sessions do not cache information on disk.

C-memory: Conventional cookies have optionally associated *paths*. A cookie with associated path P is released only when the browser in which it is resident requests a URL with prefix P. For example, a cookie set with the path "www.arbitrarysite.com/X" would only be released when the browser visits a URL of the form "www.arbitrarysite.com/X/...". Using paths, it is possible to create *C-memory*, a type of CC-memory based on conventional cookies. We can design C-memory to restrict read access to cookies based on secret keys, rather than domain names (which can be spoofed) – to the best of our knowledge, a new approach to the use of cookies. *C-memory*, like CC-memory, can support huge virtual memory structures. Our proposed protocols for CC-memory can be implemented equally well in C-memory (in systems where cookies are not blocked).

3 Schemes for User Identification and Authentication

In this section, we propose a tree-based construction called an *identifier tree* that enables a server to identify visiting users via objects stored in CC-memory. Normally, TIFs and ordinary cookies (C-memory) are *domain-tagged*, meaning that access is restricted to servers from the *domain* that set them. In a pharming attack, though, an attacker successfully spoofs a domain name and bypasses domain-based controls. Additionally, some forms of CC-memory, like that based on browser-histories, are by nature accessible to *any* server. Our identifier-tree scheme addresses these problems by restricting server access to user identifiers based on *secret keys* held by the server, instead of domains.

At the end of this section, we also briefly consider how *secret* cache cookies can aid in *authenticating* users that a server has already identified, and how they can help combat pharming attacks.

3.1 Identifier trees

On creating an identifier tree T, a server associates each of its users with a distinct leaf in the tree; nodes in the tree correspond to secrets in CC-memory. The server plants in the browser of the user the set of secret cache cookies along the path from the root to the user's leaf. To identify a visiting user, the server interactively queries the user's browser to determine which path it contains; in other words, the server performs a depth-first search of the identifier tree. In identifying the user's unique leaf, the server identifies the user. This search is feasible only for the original server that generated the identifier tree (or for a delegate), because only the server knows the secret cache cookies associated with nodes in the tree.

Consider a binary tree T. Let d denote the depth of the tree. For a given node n within the tree, let $n \parallel$ '0' denote the left child, and $n \parallel$ '1', the right child; for the root, we take n to be a null string. Thus, for every distinct bitstring $B = b_0b_1 \ldots b_j$ of length j, there is a unique corresponding node n_B at depth j. The leaves of T are the set of nodes n_B for $B \in \{0, 1\}^d$.

With each node n_B, we associate a secret value u_B, namely a secret (l-bit) address in CC-memory. To store

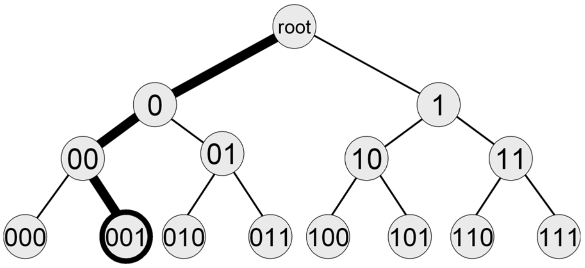

Figure 1. A simple identifier tree of depth $d = 3$ with a highlighted path for identifer '001'

node n_B in the CC-memory of a browser, a server plants a cache cookie at address u_B.

The server that has generated T for its population of users assigns each user to a unique, random leaf. Suppose that user i is associated with leaf $n_{B^{(i)}}$, where $B^{(i)} = b_1^{(i)} b_2^{(i)} \ldots b_d^{(i)}$. The server determines the leaf – and thus identity – of a user as follows. The server first queries the user's browser to determine whether it contains n_0 or n_1 in its cache; in particular, the server queries address u_0 looking for whether the corresponding bit is on or off, and then address u_1. The server then recurses. When it finds that node n_B is present in the browser, it searches to see whether $n_B \parallel$ '0' or $n_B \parallel$ '1' is present. Ultimately, the server finds the full path of nodes $n_{b_1^{(i)}}$, $n_{b_1^{(i)} b_2^{(i)}}$, $\ldots$, $n_{b_1^{(i)} b_2^{(i)} \ldots b_d^{(i)}}$, and thus the leaf corresponding to the identity of user i.

A toy, simplified identifier tree is depicted in Fig. 1. In the full paper, we discuss tradeoffs among the degree, storage requirements, and round-complexity of identifier trees.

Security of identifier trees: Space restrictions forbid in-depth security analysis of identifier trees. Our aim, however, is to protect against an adversary that: (1) Controls a number of users and thus knows their identifiers and (2) Can lure users to a rogue server via a pharming attack. We assume, however, that beyond this: (A) The adversary does not possess knowledge of the set $\{(u_B)\}_{B \in \{0,1\}^d}$ of server secrets; and (B) The adversary cannot mount an active (real-time) man-in-the-middle attack. Our scheme aims at two security goals:

1. Privacy: The adversary should be unable to extract a unique identifier for a user from her identifier tree. Consequently, the adversary should be unable on the basis of her identifier tree to establish a clear linkage between independent sessions initiated by a given user. Note, however, that the adversary can learn partial information about user identifiers and therefore can *correlate* appearances of a given user across sessions.
2. Authentication: The adversary should be unable to impersonate any user it does not control.

3.2 Secret cache cookies for authentication

Secret cache cookies can offer some resistance to pharming. A secret cache cookie is simply a secret bitstring (key) y_i specific to user i that is stored in a secret, user-specific address u_i in CC-memory (or C-memory). Secret cache cookies can act as authenticators. Once the user identifies herself and perhaps authenticates with other means, e.g., a password or hardware token, a server checks for the presence of a user-specific secret cache cookie as a secondary authenticator. We emphasize that a server gains access to the secret cache cookie *not* by merit of its domain name, but by merit of its knowledge of the secret u_i.

Security of secret cache cookies: For both restricted and unrestricted CC-memory, as well as C-memory, secret cache cookies are resistant to basic pharming. Domain spoofing, e.g., DNS poisoning, is insufficient for successful attack: In order to access the key y_i, a server must know the secret address u_i associated with a user.

A more aggressive pharming attack, however, can compromise a secret cache cookie. A pharmer can lure a user, steal her password, log into a server to learn u_i, lure the user a second time, and steal y_i. We cannot wholly defend against such a multi-phase attack, but can raise the number of required attack phases. To do so, we associate with user i not one secret cache cookie, but a sequence of d of them. A server searches for the d secret cache cookies sequentially, rejecting an authentication attempt immediately when it is unable to locate one. To defeat such an authentication scheme, a pharmer must interact with a server and client in turn at least d times. For large enough d, this virtually requires a real-time man-in-the-middle attack (or malware, which renders most authentication methods impotent).

4 Implementation

We now describe an implementation of CC-memory based on TIFs. Our server is an Apache 1.3.33 using FastCGI, Perl and Gentoo Linux (2.4.28 kernel), on a 1 GHz Pentium III with 256MB memory. Our client uses Mozilla 1.5.0.1 and Windows XP, on a machine with identical hardware as the server. Thus, the server is clearly under-powered for its task; on the other hand, we performed experiments on a 100 Mbps private local area network with minimal network traffic and congestion.

We execute a *write* to the browser cache by causing the client to make a series of HTTP requests to cacheable content. In our implementation we chose to cache GIF image files referenced from a dynamically generated document. These images contain solely the HTTP header and no actual content, resulting in very quick loads. The HTTP/1.1 server response header for the first load contains *Last-Modified*, *ETag*, *Cache-Control*, and *Expires* fields and values. The Cache-Control and Expires fields are set to instruct the Web client to cache the content many years into the future. An ETag (short for "entity tag") is a field that enables a server to distinguish among different instances of a single resource, e.g., different versions or copies of a cached browser image.

We execute a *read* via subsequent client retrievals of the cached objects. These result in the client sending *Last-Modified* and *ETag* values to the server in HTTP requests in the form of *If-Modified-Since* and *If-None-Match* fields respectively. If these values match those in the initial *write*, then a cache hit is observed. In this case, the server returns an HTTP 304 (Not Modified) response so as not to "clobber" the cached value. Otherwise, it returns a 404 (Not Found) HTTP response. (This process of a client sending data to a server to be validated is called a conditional GET request.)

Our uses proposed above for cache cookies are likely to involve considerably more frequent reads, i.e., authentications, than writes, i.e., initializations. Thus in our experiment we measured the full, round-trip time for the server to read a batch of n TIFs, i.e., to read n TIFs in a single communication round. We refer to Figure 2 for our results; we have plotted one hundred data points for each value of n within the range of 1 to 80.

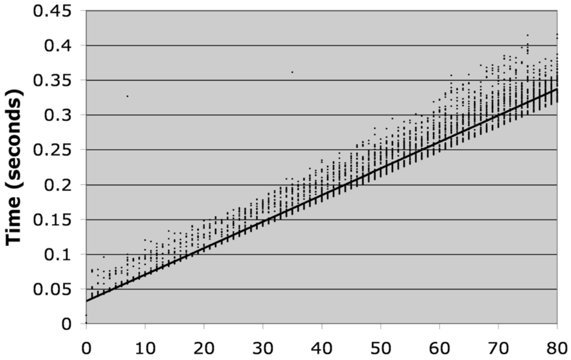

Figure 2. Round-trip time for a server to read a batch of n TIF cache-cookies

As an example, consider a translation of these timing results into a performance estimate for an identifier tree, such as a binary tree of depth $d = 60$. For $n = 2$, the average read time was 0.04175 seconds. This corresponds to the expected time for the server to test the pair of descendants of a given node. Thus traversal of the full tree would require an average of approximately 2.5 seconds.

We can greatly extend the amount of information in a TIF in CC-memory by co-opting two fields. There is the *Last-Modified* field, which contains 32 bits. The *ETag*, though, is particularly useful for our purposes; in Mozilla 1.5.0.1, for example, an ETag can contain up to 81864 bits. (The line buffer for the ETag is 10k bytes, some devoted to header information.) Thus for secret cache cookies, a single TIF can furnish essentially as much secret data as needed – well beyond the 128 bits typical for a cryptographic secret key.

5 Conclusion

We have shown that careful deployment, cache cookies can support privacy-sensitive user identification. They can also strengthen user authentication and help protect against phishing and pharming attacks. Users are increasingly supressing cookies because of privacy concerns. Cache cookies are an alternative that can replace some of the resulting, lost functionality. Additionally, the techniques we have introduced for cache cookies can be applied to ordinary cookies to help strengthen their use in authentication.

For further details on our work, we refer to the full paper, available at www.ravenwhite.com.

References

[1] A. Clover. Timing attacks on Web privacy (paper and specific issue), 20 February 2002. Referenced 2006 at www.securiteam.com/securityreviews/5GP020A6LG.html.

[2] E. W. Felten and M. A. Schneider. Timing attacks on Web privacy. In *ACM Conference on Computer and Communications Security*, pages 25–32. ACM Press, 2000. Referenced 2006 at http://www.cs.princeton.edu/sip/pub/webtiming.pdf.

[3] C. Jackson, A. Bortz, D. Boneh, and J. Mitchell. Web privacy attacks on a unified same-origin browser. In *WWW 06*, 2006. To appear.

[4] J. Vijayan. Microsoft warns of fraudulent digital certificates. *Computerworld*, 22 March 2001. Referenced 2006 at www.computerworld.com/softwaretopics/software/story/0,10801,58857,00.html.

Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)
1081-6011/06 $20.00 © 2006 **IEEE**

IEEE
COMPUTER
SOCIETY