Journal of Internet Services and Applications
a SpringerOpen Journal

RESEARCH Open Access

Secure cross-domain cookies for HTTP

Paul Rabinovich

Abstract

Cookies represent an important element of HTTP providing state management to an otherwise stateless protocol. HTTP cookies currently in use are governed by the same origin policy that directs Web browsers to allow cookie sharing only between Web sites in the same DNS domain. As Web applications get richer, data sharing across domain boundaries becomes more important. While practical solutions to cross-domain data sharing exist, in many cases they increase complexity and cost. In this paper we propose a simple mechanism to share cookies using authorizations based on X.509 attribute and public key certificates. In addition to supporting secure cookie sharing between unrelated domains, it can be beneficial for hosts in the same domain when the currently used same origin policy is deemed too permissive, exposing cookies to leakage and spoofing.

Keywords: HTTP, Cross-domain cookie, Public key cryptography, Authorization model

1 Introduction

An HTTP cookie is a small file left in a Web browser by a Web server. Cookies were introduced by Netscape as a state management mechanism to the otherwise stateless HTTP protocol in 1994 [1]. The current implemented standard for HTTP cookies is RFC 2109 [2]; it defines the cookie format and the rules for proper handling of cookies by Web browsers, servers, and proxies. A newer standard, RFC 2965 [3], was never widely adopted[a]. A cookie carries a name/value pair (its useful payload) and a set of management attributes. The basic management attributes are shown in Table 1.

Cookies may be subdivided into *session cookies*, erased when the browser is closed, and *persistent cookies*, preserved across multiple browser sessions. Persistent cookies are frequently used as an inexpensive way to store users' preferences for a Web site. The advantage is that the Web site operator doesn't need to maintain an account for the user; all pertinent information is stored in the browser, and made available to the site when the user visits it. Cookies may also be used for tracking purposes: a cookie lets a Web site uniquely identify the user (on a given computer) and link to her all Web pages she visited on that site. Tracking is possible within a single Web site or across a group of cooperating Web sites.

A Web page is a hypertext document more often than not referencing other resources on the Web. Cookies received by the browser when accessing these resources are frequently called *third-party cookies*; those set by the main page are called *first-party cookies*. This distinction does not affect the HTTP protocol. It reflects the end users' perception of these artifacts and has important implications for user privacy since third-party cookies are frequently used for user tracking across multiple sites. See [5] for more details.

Browsers use the *same origin* policy to determine whether to send a cookie to a Web site: an HTTP request sent to a host will contain those and only those cookies whose Domain attribute identifies the host itself or the DNS domain to which the host belongs [2][b]. (The Path and Port attributes are also taken into account.) When setting a cookie, the Web server is allowed to omit the Domain attribute (then the browser sets this attribute to the server's host name) or to set it to the server's parent domain. For example, `host x.domain1.com` may set Domain to `.domain1.com` but not to `.domain2.com`. To set cookies Web servers use the `Set-Cookie` HTTP header; to relay cookies to Web servers browsers use the `Cookie` header[c].

One of the requirements of the same origin policy is that cookies be shared only between Web sites within the same *administrative* domain. Various heuristics were put in place to guess administrative domain boundaries using the hierarchical structure of DNS names [1,2,6]. For instance, servers cannot set the Domain attribute to one of the top level domains (TLD) since by definition each

Correspondence: paul.rabinovich@exostar.com
Security Software Development, Exostar, Herndon, USA

Table 1 HTTP cookie attributes [2]

Attribute	Description
Comment	Short description of the intended use of the cookie
Domain	DNS domain or IP address for which the cookie is valid
HttpOnly	If present, the cookie cannot be accessed by a client-side script (e.g., written in JavaScript). Although non-standard, this attribute is supported by most Web browsers [4]
Max-Age	Maximum period after which the cookie must be discarded
Path	Subset of URLs on qualifying hosts for which the cookie is valid
Port	List of TCP ports on qualifying hosts for which the cookie is valid
Secure	If present, the cookie may be transported only over a secure (e.g., SSL-protected) channel

TLD provides a common umbrella for a (large) set of completely unrelated domains.

The original Netscape proposal [7] never included provisions for cookie sharing between arbitrary Web sites. The subsequent standards [2,3] didn't change this basic assumption. In many cases, however, the same origin policy imposes unnecessary limitations on Web site developers and forces them to implement complex and expensive workarounds. They are also prone to failure. For example, sharing of state information can be accomplished by embedding data in URLs or posting them via HTML forms. But if the user clicks the Back button, the application's state will be rolled back rendering the user's experience inconsistent [1].

While the Web technology itself imposes limitations on state and context sharing across domain boundaries, the general trend in Web development is towards increased integration, regardless of such boundaries. Federated Web portals, Web mashups and composite applications provide a unified experience to end users, combining data, resources, and elements of the user interface from multiple sources [8-10]. Sharing of persistent data within these systems remains a challenge. As one example, in the world of identity federation it's often useful to automatically discover a user's identity provider, or IDP (an entity that holds her account and authenticates her) when she visits an affiliated Web site (a service provider, or SP). One way to accomplish this is for the identity provider to leave a cookie in the user's browser; subsequently the cookie can be read by service providers, and the browser can be automatically redirected to the IDP for authentication. Indeed, the SAML 2.0 Identity Provider Discovery Profile [11] uses this approach but necessarily assumes that the IDP and the SP(s) share a DNS domain. This model works well when used by a small number of closely related sites but is not practical for large-scale identity federations [12].

Recognizing stultifying effects of the same origin policy, manufacturers of Web client software (e.g., Adobe Flash, Microsoft Silverlight) added support for cross-domain communication (including cookie sharing) to their products, and recent analyses demonstrate that the use of this feature is on the rise [13,14]. However, there is no comparable *standards-based* mechanism for cross-domain cookie sharing.

The same origin policy for cookies as it is currently implemented may be too permissive in some cases. For example, some country-code top level domains have second level subdomains that act as generic, functional top level domains in their respective hierarchies. In many cases current domain name matching rules allow sharing of cookies with all sites in such domains [6,15]. Even in domains with more "administrative affinity" some hosts may want to interact via cookies without necessarily sharing information with, or receiving information from, other peer hosts. This unwanted sharing may result in cookie leakage (cookies are sent to unauthorized Web servers) or cookie spoofing (cookies are inadvertently or maliciously set by unauthorized Web servers) [1].

From these examples it should be clear that cookie sharing across domain boundaries is a desirable feature in Web applications and middleware. For such sharing to be secure an authorization mechanism needs to be developed granting access only to those hosts that require it. As our last example indicates, even hosts within a single domain may benefit from a more fine-grained access control model than the one currently in use. In this paper we propose such a model and introduce modifications to the HTTP protocol necessary to support it.

This paper is organized as follows. Section 2 provides a review of related work. Data structures required by our solution are described in Section 3. Section 4.3 explains how cross-domain cookies are set, and Section 4.4, how they are read. In Section 6 we discuss our proof of concept and evaluate communication overhead of the proposed solution. In Section 7 we analyze our proposal focusing especially on its security properties and implementability. Lastly, Section 8 summarizes the paper.

2 Related work

Many variants of the same folk protocol exist where one or more Web sites use another site as a cookie manager (CM): to set a cookie the Web sites redirect the browser to the manager passing the necessary data as a request parameter; the manager sets the cookie when redirecting back to the requesting page. (At this point the browser associates the cookie with the cookie manager's host or domain.) To receive a cookie, they do a redirect to the CM who receives the cookie from the browser and performs another redirect (back) passing the data, also as a request parameter. While this is a working solution, it requires

multiple redirects (two per request) increasing communication overhead and design complexity. Our proposal, on the other hand, introduces native support for cross-domain cookies in HTTP and, thus, eliminates the need for redirects.

Callaghan et al. proposed a proxy-based solution that allows non-cooperating Web servers to communicate using standard HTTP cookies [16]. A forwarding proxy is configured to treat a group of Web sites as one; it captures cookies from passing HTTP traffic and makes them available to communicating browsers and servers by inserting `Cookie` and `Set-Cookie` HTTP headers as needed. In a more complex configuration, Callaghan et al. set up a "cookie manager" URL that the proxy itself responds to. Cross-site cookies are associated in the browser with that URL. When a browser sends a request to a participating server, the proxy initiates a cookie transfer from the "cookie manager" (CM) as follows:

1. Redirect to the CM keeping the target URL as a request parameter.
2. Intercept the request as the CM, receive all cookies, and redirect to the target URL encoding the cookies as request parameters.
3. Intercept the request again, convert the request parameters into cookies using the `Set-Cookie` header, and redirect to the target URL again.

After this the cookies will be associated with the target host as well as the CM itself. The drawback of this solution (in addition to multiple redirects) is that it tightly couples components in the user's domain (the forwarding proxy) and the application's domain (the Web servers). Such coupling may be achieved and maintained in a controlled environment, for example, within an enterprise, but cannot be easily replicated in other settings. Callaghan et al.'s approach was driven by constraints imposed by the same origin policy. Our solution does not adhere to this policy, cross-domain cookies are supported natively, and no additional components (such as proxies) are required. Since the proxy is not needed any longer, the tight coupling is eliminated, and the operational complexity of the system, reduced.

Guo and Zhou proposed a new type of cookie (called a *cross cookie*) geared towards Web mashups [17]. Their model consists of three tiers: content servers serving *gadgets*, aggregating servers that combine gadgets into mashups, and, finally, Web browsers rendering mashup pages. Cross cookies are exchanged between aggregating servers and browsers. A mashup can be represented as an HTML document object model (DOM) tree with subtrees containing gadgets from content servers. When an aggregating server receives a cookie with a gadget, it constructs a cross cookie capturing the name of the content server and the position of the gadget in the mashup DOM tree, and sends the cookie to the browser. Similarly, the browser sends cross cookies back to the aggregating server, and the latter converts them to traditional cookies when communicating with the content server-owner of the gadget. For example, if the same parameterized gadget is included k times in a mashup, the content server may send as many as k versions of the same cookie to the aggregating server; cross cookies will capture the cookie context (the position in the DOM tree) and provide enough information to the aggregator to return the correct version of the cookie on subsequent visits to the content server. Despite their name, cross cookies are not general purpose cross-domain cookies; they address a special need of Web mashup applications and work only between aggregating servers and browsers. By contrast, our proposal is general in nature and can be used in any HTTP-based communications.

The HTML 5 specification introduced *Web Storage*, a new state management mechanism for the Web [18]. The standard supports session storage and persistent storage, both indexed by origin. A storage object is a simple associative array. Although stored objects are governed by the same origin policy, they can participate in cross-domain data sharing through the use of another HTML 5 mechanism, *Web Messaging* [19]. Web Messaging is a JavaScript API for data exchange between browser windows. The communicating parties (scriptlets) may belong to different origins; they themselves make the decision whether to send (receive) a message based on the recipient's (sender's) origin. The significant advantage of the Web Storage/Web Messaging combination is its acceptance by the market. (Although [18,19] are only draft specifications, they have been implemented by all major browsers [6].) It requires careful client-side coding, however. Although both senders and recipients are encouraged to check each other's origins [20], in practice this is done inconsistently. For example, Hanna et al. [21] discovered that two of the most popular users of Web Messaging, Facebook Connect and Google Friend Connect, perform these checks only sporadically; the authors were able to compromise message integrity and confidentiality of both protocols. Our approach, on the other hand, does not require any coding; it abstracts security decisions into a small set of simple data structures (channel and authorization certificates) that lend themselves to efficient unified management by Web sites.

Another widely available cross-domain data sharing mechanism is *Cross-Origin Resource Sharing* (CORS) [22]. CORS allows a Web page associated with one origin to access resources associated with a different one. Based on the `Origin` header reported by the browser, the target Web site may choose to allow or deny access, or, more granularly, accept or expose certain HTTP headers (including `Cookie` and `Set-Cookie`). Like Web

Storage/Web Messaging, the CORS specification is supported by all major browser vendors. Its drawback is that it can only be used with AJAX (JavaScript) requests [23]. Our cross-domain cookies work like traditional cookies; they can be used with both browser-native and JavaScript-issued HTTP requests[d].

3 Channels, cookies and authorizations

3.1 Channels and channel names

To allow disparate domains to communicate using cookies we introduce the notion of a *cross-domain channel* (XDC). An XDC channel may be thought of as a folder in the browser to which writers write cookies and from which readers read them. Cross-domain channels have names. We propose a decentralized namespace where owners create and destroy channels as needed without coordinating it with anybody else. To avoid collisions we use channel names based on RSA keys. When creating an XDC channel, its owner generates a random RSA key pair (with a sufficiently long modulus), and computes a digest of the public key using a high quality hash algorithm. The computed digest is the channel name. This approach allows us (a) to generate names that are unique for all practical purposes, and (b) to use a simple scheme to prove one's ownership of a channel: just present the public key from which the channel name was derived and prove possession of the private key corresponding to it.

Channel owners issue *XDC channel certificates* to themselves and *XDC authorizations* to Web sites or DNS domains interested in using their channels. These data structures are covered in Section 3.3.

3.2 Secure channels

The owner may designate an XDC channel as secure. XDC cookies associated with a secure channel may be transmitted only over a secure (e.g., SSL-protected) connection. This is similar in spirit to the Secure attribute in traditional cookies [2]. Transmitting cookies only over a secure transport has several benefits. It enhances security and confidentiality of the cookies themselves. It also helps to mitigate against DNS spoofing: when a browser validates a server's certificate, it verifies that the host name in the certificate matches that of the host the browser is actually communicating with. To fool the browser the attacker would have to spoof the DNS and to gain access to the server's private key. In addition, in our scheme (Section 3.3) XDC authorizations for secure channels are bound to the holder's SSL certificate, providing an extra layer of protection against Web site impersonation.

Internet attribute certificates already have a provision for referencing the subject's identity certificate (specifically, its issuer name and serial number) [24]; we reuse this mechanism. When a Web browser receives a cookie, it finds the cookie's authorization; if it contains a reference to the server's identity certificate and the cookie was received over an insecure channel, the cookie is ignored; if it's received over an SSL channel, the browser checks if the server's SSL certificate matches the reference in the XDC authorization: on match the cookie is accepted, on mismatch, discarded. XDC cookie delivery to Web servers works in a similar fashion. This is analogous to the approach proposed by Karlof et al. for traditional cookies [25].

3.3 Channel certificates and authorizations

The owner of an XDC issues authorizations to hosts that need to use it. The structure of an XDC authorization is shown in Figure 1. It consists of two components, an optional channel certificate and an authorization certificate granting access to the channel to a particular host or DNS domain.

The channel certificate is a self-signed X.509 public key certificate [26,27]. It contains a human-readable description of the channel, the channel name (along with the identifier of the hash algorithm used to compute it), and the secure flag. It is signed with the owner's private key. The channel certificate is optional in an authorization. If it is not included, the relying party (e.g., the browser) must be able to obtain it by other means. Our scheme provides a discovery mechanism for doing it.

The authorization certificate is also signed with the owner's private key. It is implemented as an Internet attribute certificate defined in RFC 3281 [24]. The authorization's subject's name is placed in the Holder field of the certificate. An authorization grants its holder a permission to read or to write (i.e., create, modify and delete) XDC cookies. In addition to the holder's name and permissions, the authorization certificate encodes the XDC channel name and a brief human-readable description of the channel or of the holder's use of the channel. For secure channels it also contains the ID of the holder's SSL certificate (the issuer name/serial number pair [24]). Like any attribute certificate, the authorization certificate has a validity period that must be checked every time the corresponding XDC cookie is used.

Internet attribute certificates support a revocation checking model based either on certificate revocation lists (CRL) or on the Online Certificate Status Protocol (OCSP). To simplify processing and minimize the overhead, however, we chose not to use revocation checking in our solution.

3.4 Cross-domain cookies

Cross-domain (or XDC) cookies have essentially the same structure as HTTP cookies currently in use. They still carry name/value pairs and additional management attributes (see Table 1). Instead of the Domain, Path and Port attributes, however, we introduce the XDC Name

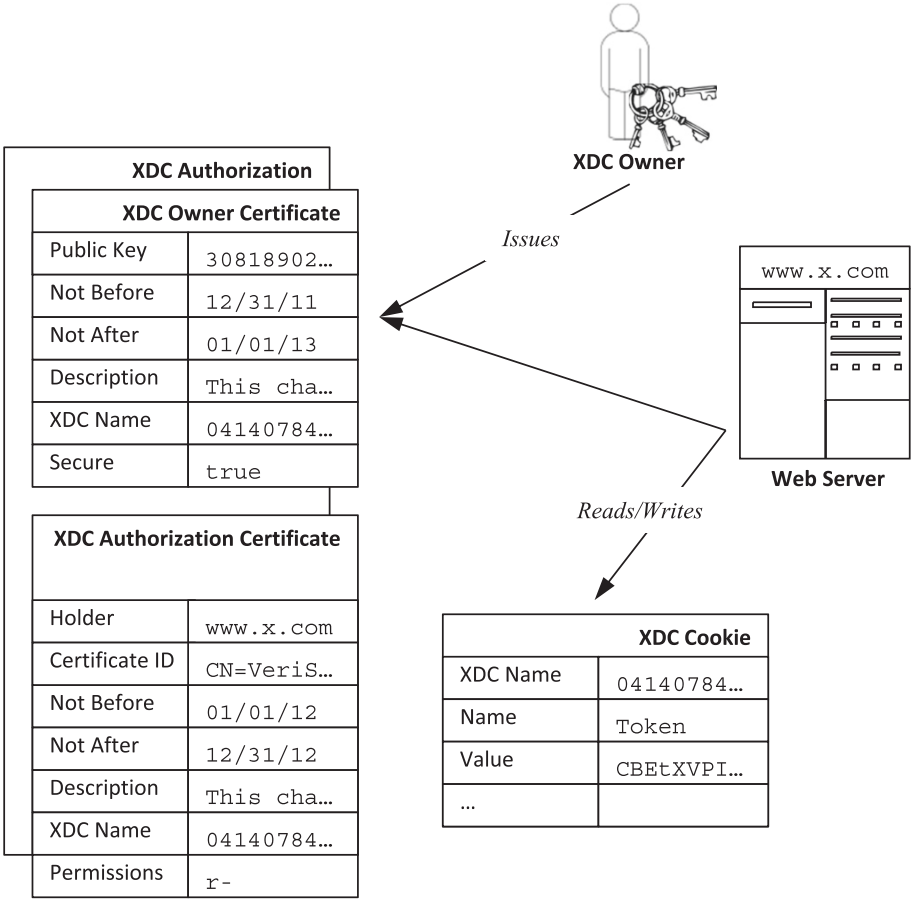

Figure 1 Channels, cookies and authorizations.

attribute; it contains the (properly encoded) name of the XDC channel to which the cookie belongs. XDC cookies do not allow access to scripts, so the HttpOnly attribute is not required, either. Finally, the Secure attribute is superseded by the (more resilient) secure flag in the XDC channel certificate.

4 The protocol

4.1 Data exchanges

When the browser makes a user-initiated request to Web server, it may have some XDC cookies to send to the server subject to the server's XDC authorizations. The browser may have some, all, or none of the authorizations issued to the server (or its parent DNS domain). Our proposal provides three mechanisms to discover all applicable authorizations:

- Send a *preflight request* to the server
- Send the user request, allow the server to provide any missing authorizations, and then resend the request again
- Perform a DNS lookup

We issue preflight requests for all user requests that use HTTP methods POST and PUT. Not doing so may result in the server activating the second discovery mechanism which, in turn, may lead to retransmission of large amounts of data in the request. Preflight information received from the server is cached (for the duration indicated by the server), so not all user requests require preflight authorization; only those with expired (or non-existent) cache entries do. Preflight information may include XDC channel certificates and XDC authorizations as well as the time to live for the information and additional options. The only option currently defined is a flag indicating whether the browser should use the DNS to look up additional authorizations for the server (or its parent DNS domain) and, if so, how often.

When the server receives an actual user request, it may discover that some XDC cookies it expects have not been included by the browser: the browser either doesn't have the cookies at all, or it has the cookies but doesn't have some authorizations for the server. To account for the latter case, the server may respond with missing authorizations, and request the browser to repeat the request,

now with the missing cookies (presumably, covered by the just discovered authorizations).

A browser receiving XDC cookies from a server can use all previously cached authorizations or authorizations the server sends with the cookies themselves. Since the server can always bundle cookies and authorizations in a single response, no additional round trips are required to complete discovery.

The last source of XDC authorizations for a Web site is the DNS. To support out-of-band delivery, we propose to place XDC authorizations in TXT resource records[e] (RR) in the DNS [28], encoded to respect the rules of the DNS. Since TXT resource records may be used by many applications, there is a risk that a record received by the browser is not an XDC authorization. In our implementation the client just discarded the RR if it could not interpret it correctly; to minimize unnecessary traffic in the future an application-specific RR could be introduced or a general-purpose "kitchen sink RR" [29] reused if one is implemented.

A DNS lookup is performed on send when the browser finds an unresolved XDC cookie in the cookie "jar", and on receive when the server sends an XDC cookie that cannot be resolved by any other means (cached or in-band XDC authorizations). The DNS is not consulted unless the server indicates it in its response to a preflight request.

4.2 Preflight requests

A preflight request is an HTTP request; it uses the HTTP method OPTIONS and sets the request header `Xdc-Info-Request` to `true`. A preflight request is issued for the same URL as the original user request. A compliant server may return zero or more `Xdc-Channel` and `Xdc-Authorization` headers. It may also include the `Xdc-Max-Age` header indicating the maximum retention time of the information provided in the response. (If none is given, a protocol default will be used.) Finally, an `Xdc-Options` header may include additional XDC processing instructions; currently only the `dns-max-age` option is defined; if set, it instructs the browser to look up missing XDC authorizations in the DNS and defines the maximum frequency of such lookups. A sample preflight request is shown in Figure 2.

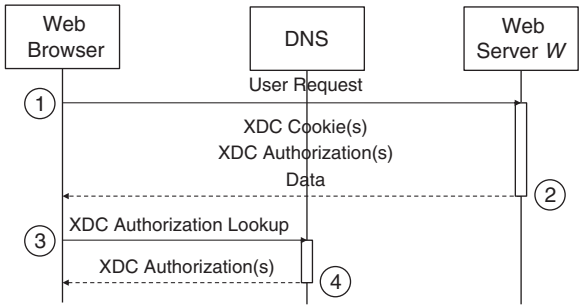

Figure 3 The sequence diagram for setting of XDC cookies. The browser sends the user's request and receives a response that may contain XDC cookies and authorizations. If any of the XDC cookies don't have a matching authorization, a DNS lookup is performed.

4.3 Setting cross-domain cookies

Figure 3 shows the sequence diagram for an XDC cookie-setting server. The server uses the `Xdc-Authorization` (and possibly the `Xdc-Channel`) headers to convey its authorizations to the browser. The value of the header is an encoded XDC authorization. We use double encoding: first the value is base 64-encoded and then URL-encoded. To set cross-domain cookies our server uses the new `Xdc-Set-Cookie` header. Normally, the client would discard those cookies for which the server failed to provide an authorization. In our model, however, the client may contact the DNS to retrieve the missing authorizations. All XDC authorizations in hand, the client validates the cookies and stores them in the cookie "jar".

4.4 Reading cross-domain cookies

The sequence diagram for an XDC cookie-reading scenario is shown in Figure 4. Before sending XDC cookies

```
>> OPTIONS /xdc/read HTTP/1.1
>> Host: read.xdc.com:8080
>> Xdc-Info-Request: true
>> Connection: Keep-Alive
>> User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:7.0.1) Gecko/20100101 Firefox/7.0.12011-10-16 20:23:00

<< HTTP/1.1 200 OK
<< Server: Apache-Coyote/1.1
<< Transfer-Encoding: chunked
<< Content-Length: 0
<< Xdc-Channel: MIIB%2FzCCAWigAwIBAgIBATANBgkq...
<< Xdc-Authorization: MIIEB6CCAgMwggH%2FMIIBaKADAgEC...
<< Xdc-Max-Age: 36000
<< Xdc-Options: dns-max-age=1000
<< Date: Tue, 27 Dec 2011 19:15:01 GMT
```

Figure 2 A preflight request. **>>** marks headers sent to the server, **<<**, headers received from it.

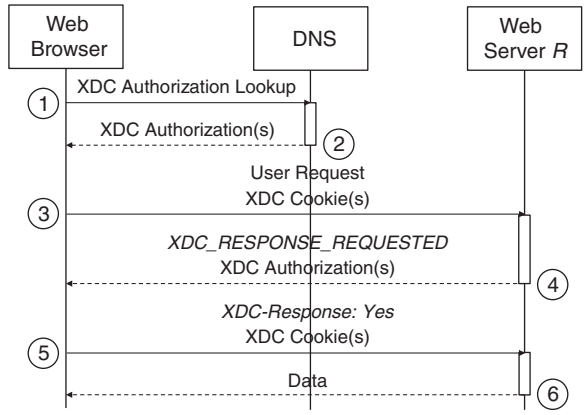

Figure 4 The sequence diagram for reading of XDC cookies. Prior to sending the user's request, the browser looks up missing authorizations (if any) in the DNS. Then the request is sent along with XDC cookies. The server may provide additional authorizations in order to receive XDC cookies it expected but did not receive (step 4). Having received an XDC response in step 5, the server responds with data. Steps 4 and 5 should not be needed for user requests requiring preflight authorizations.

to the server the Web client needs to find all missing authorizations. It contacts the DNS and requests TXT resource records for the Web server's host. Having received and validated the authorizations, it sends the appropriate XDC cookies to the server using the new `Xdc-Cookie` header. As we mentioned, the server may expect additional XDC cookies that it doesn't receive with the request. Our Web server may respond with the missing `Xdc-Authorization` headers and set the HTTP status code to `XDC_RESPONSE_REQUESTED`, a value we introduced. This status code tells the browser that the sole purpose of the HTTP response is to provide the necessary XDC authorizations and that the browser must repeat the request including all valid XDC cookies. Since now two requests are treated as a single request, the server needs to remember that it already provided all XDC authorizations it has. To avoid the need to store the state of this two-step request on the server we propose a new header, `Xdc-Response` with values `true` and `false`. When a client repeats a request in response to the `XDC_RESPONSE_REQUESTED` status code, it sets this header to `true` and includes it in the request. Not sending the header is equivalent to sending `Xdc-Response: false`. The client repeats the operation by evaluating all XDC cookies it holds against the updated set of XDC authorizations for the target host.

Figure 5 shows the trace of a single request to an XDC cookie-reading server. Initially the browser doesn't have any authorizations for host `read.xdc.com`. The server responds with a set of four authorizations and sets the HTTP status code to `XDC_RESPONSE_REQUESTED` (399). The browser repeats the request setting the `Xdc-Response` header to `true` and including all eligible cookies. There are six valid cookies for the host spread over three cross-domain channels.

```
>> GET /xdc/read HTTP/1.1
>> Host: read.xdc.com:8080
>> Connection: Keep-Alive
>> User-Agent: Apache-HttpClient/4.1-alpha1 (java 1.5)

<< HTTP/1.1 399
<< Server: Apache-Coyote/1.1
<< Xdc-Authorization: MIIC2DCCAZkwggECoAMCAQICBEs%2B...
<< Xdc-Authorization: MIIC3DCCAZkwggECoAMCAQICBEtXVP...
<< Xdc-Authorization: MIIC4TCCAZkwggECoAMCAQICBEtXVP...
<< Xdc-Authorization: MIIC4TCCAZkwggECoAMCAQICBEtXVi...
<< Content-Length: 0
<< Date: Tue, 27 Dec 2011 18:28:43 GMT

>> GET /xdc/read HTTP/1.1
>> Xdc-Response: true
>> Xdc-Cookie: name=A3; value=Value3; channelName=axual9nhqawkv2ejkwwgzuig0q4xwtq%3d
>> Xdc-Cookie: name=B3; value=Value3; channelName=axual9nhqawkv2ejkwwgzuig0q4xwtq%3d
>> Xdc-Cookie: name=A1; value=Value1; channelName=axua9a7s5o87a7pasuu2xvbcyzsq1gc%3d
>> Xdc-Cookie: name=B1; value=Value1; channelName=axua9a7s5o87a7pasuu2xvbcyzsq1gc%3d
>> Xdc-Cookie: name=A2; value=Value2; channelName=axuafo%2bx2rcz4qke%2fkccqh8mustitos%3d
>> Xdc-Cookie: name=B2; value=Value2; channelName=axuafo%2bx2rcz4qke%2fkccqh8mustitos%3d
>> Host: read.xdc.com:8080
>> Connection: Keep-Alive
>> User-Agent: Apache-HttpClient/4.1-alpha1 (java 1.5)

<< HTTP/1.1 200 OK
<< Server: Apache-Coyote/1.1
<< Transfer-Encoding: chunked
<< Date: Tue, 27 Dec 2011 18:28:43 GMT
```

Figure 5 An HTTP request relaying XDC cookies to a server. >> marks headers sent to the server, <<, headers received from it.

4.5 Summary of changes to the HTTP protocol

In this section we summarize all additions to the HTTP protocol required to support our cross-domain cookies. Table 2 lists our proposed HTTP headers. Table 3 lists the single newly proposed HTTP status code.

5 Proof of concept

Our proof of concept consists of several components:

- Utilities to generate channel certificates, sign XDC authorizations, and save them in different formats
- An XDC cookie-reading application
- An XDC cookie-writing application
- An HTTP server capable of servicing XDC preflight requests
- A DNS server configured with XDC authorizations
- An XDC-capable client

The utilities and applications were written in Java. We used Bouncy Castle's cryptographic APIs [30] for all work with X.509 attribute and public key certificates. To host the applications, we used Apache Tomcat [31]. We enabled server-side support for preflight authorization by configuring our Apache HTTP proxy with two modules, mod_rewrite [32] and mod_headers [33]. The mod_rewrite configuration set an environment variable on HTTP OPTIONS requests that contained the header `Xdc-Info-Request: true`; the mod_headers configuration output XDC-specific HTTP headers if the environment variable was set. The DNS server hosted XDC authorizations. We used BIND 9.7.0 [34]. Our XDC-capable client was implemented as a Firefox browser extension. Firefox provides a pluggable framework for extending its functionality, and a cross-platform component object model, called XPCOM, for programming the extensions [35]. Multiple language bindings are supported for XPCOM; we implemented our extension in JavaScript. All source code and configuration instructions for our proof of concept are available from [36].

6 Evaluation

Our proposal adds communication overhead to normal browser/Web server interactions. Web sites not aware of our cross-domain cookies will incur minimal cost: the initial preflight request will either fail or return no information, and a protocol default (several days) will define the frequency of subsequent requests; DNS lookups will not be issued (no explicit instructions in the preflight response); and repeat requests will never be initiated. Overhead imposed by XDC-aware Web sites will depend on:

- The frequency of preflight requests
- The frequency of DNS lookups
- The frequency of repeat requests
- The number of channels with which the site interacts (i.e., reads or writes XDC cookies)
- The size of an individual XDC channel certificate and XDC authorization
- The number and size of XDC cookies set and received

Web sites can control the frequency of preflight requests by setting the header `Xdc-Max-Age`, and the frequency of DNS lookups by setting the header `Xdc-Options` (Section 4.2). Both settings are subject to tradeoff analysis

Table 2 HTTP headers

Name	Description	Request/response	When used
Xdc-Authorization	Contains a safely-encoded XDC authorization	Response	User, preflight requests
Xdc-Channel	Contains a safely-encoded XDC channel certificate	Response	User, preflight requests
Xdc-Cookie	The short representation of an XDC cookie (similar to the standard `Cookie` header [2]). May be replaced by the standard header if the Domain attribute can be overloaded with the XDC name and used included in requests (currently not supported)	Request	User, repeat requests
Xdc-Info-Request	Indicates a XDC preflight request. Always set to `true`	Request	Preflight requests
Xdc-Max-Age	Indicates the maximum retention period for preflight information	Response	Preflight requests
Xdc-Options	Provides additional instructions for XDC processing. Currently only the `dns-max-age` option is defined	Response	Preflight requests
Xdc-Response	Set to `true` to indicate a repeat request	Request	Repeat requests
Xdc-Set-Cookie	The long representation of an XDC cookie (similar to the standard `Set-Cookie` header [2]). May be replaced by the standard header if the Domain attribute can be overloaded with the XDC name	Response	User, repeat requests

Table 3 HTTP status codes

Mnemonic name	Description	Value	When used
XDC_RESPONSE_REQUESTED	Indicates that the Web server is expecting additional XDC cookies and, assuming the browser is missing XDC authorizations, for them sending some or all of them in the response	399	User requests (to initiate a repeat request)

(the number of unnecessary requests that discover no new information against the latency of discovering a change) but in most cases they can be set to days, weeks and even months.

It can be recalled from Section 4.4 that Web servers may initiate repeat requests only when XDC cookies they expect to receive are not provided by the browser. We expect that after several initial communications the browser will have all authorizations for a given Web server, and additional exchanges will not be required. Web servers catering to low bandwidth clients may elect to store all their XDC authorizations in the DNS; Web clients will only consult the DNS when an authorization for a particular cookie is missing.

Analysis by Tappenden and Miller [37] shows that the average number of cookies used by Web sites is 2.92, and the median number is 1.0; 75% of all sites use four or fewer cookies. This suggests that the number of cross-domain cookies used by a typical Web site should be small, and the number of channels with which they are associated, even smaller. About 278 kB of data are transferred in an average Web application session [38]. In our prototype fully-encoded XDC authorizations varied between 1,108 and 1,440 ASCII characters for 1024-bit RSA keys [f]. Even if preflight authorizations are not used, only a relatively small amount of data will be added to each session.

Our headers `Xdc-Cookie` and `Xdc-Set-Cookie` have the same basic structure as the corresponding headers for traditional cookies. As we explained in Table 2 the traditional headers can even be overloaded to support XDC functionality. Assuming that the payload size and the number of cross-domain cookies and of traditional cookies will not significantly differ, any additional overhead may come only from the XDC Name attribute. Based on the length of a raw XDC name (160 bits for SHA-1-generated names) and the fact that we use double encoding (base 64 and URL), it can be shown that the average length of an encoded XDC name is 31.6 characters. (In the interest of brevity we omit the calculation.) Comparing `Xdc-Set-Cookie` to `Set-Cookie` (which may carry a Domain attribute), and `Xdc-Cookie` and `Cookie` (which may not), we get the worst case average difference of 31.6 characters per cookie.

7 Discussion

7.1 General

The proposed scheme has several important properties. First, it allows us to generate unique channel names with negligible probability of collisions. Second, XDC authorizations provide a simple access control mechanism roughly equivalent to the one currently in use on the Web based on the domain matching rules and the same origin policy. Indeed, a traditional Web client looks at the Domain attribute in a cookie and decides if the communicating Web server's host name matches it; veracity of the host name is ascertained using the DNS. In our case, host name matching is based on direct comparison of the host name as reported by the DNS and the host name in the XDC authorization. Third, an XDC authorization is unforgeable (with current technologies); it cryptographically binds permissions to the cross-domain channel name which in turn is cryptographically bound to the owner of the channel: only the owner, possessor of the private key, could have signed the authorization. The binding between the owner's public key and the channel name relies on collision resistance properties of the hash function used to compute the name of the channel. In our experiments we used the SHA-1 algorithm [39].

7.2 Trust model for authorizations

As discussed in Section 3 our scheme does not support authorization revocation. Once granted, an XDC authorization remains valid until it expires, and cannot be withdrawn. In addition, if the XDC owner's private key is compromised, there is no remedial mechanism in place to migrate to the use of a new key (and a new XDC). We argue that this risk is acceptable. In the unlikely event that the key is compromised, the owner can generate a new key pair thus creating a new XDC, issue new XDC authorizations, and distribute them to all participating Web servers out-of-band. Servers that read XDC cookies can stop accepting the old cookies right away even if browsers continue to store them (and the old XDC authorizations) until expiration. Since anecdotal evidence suggests that revocations of SSL certificates due to key compromise are extremely rare, we expect that revocations of XDC keys will be infrequent as well. Building a complex infrastructure for such rare events, in our view, is not warranted.

Our use of the public key infrastructure in XDC authorizations is somewhat unconventional. The owner of a cross-domain channel acts as an application-specific certification authority (CA) whereas under normal circumstances CAs are application-independent, although they may issue end entity certificates suitable for a particular application. Using the traditional approach would have made our cookies less lightweight since (a) more information would need to be carried in XDC authorizations and (b) another trust infrastructure[g] would have to be tapped into to validate the application-specific certificates issued to XDC owners. As it is, our trust infrastructure is self-contained and doesn't have any external dependencies.

7.3 Threat model

A cookie can be viewed as a passive data element that interacts with the following actors: Web sites, the network, the browser, and the user. Traditional cookies are built on a *user-centric* threat model. The same origin policy assumes that the user is the ultimate owner of a cookie. If the browser and the network are honest, it protects against dishonest (or curious) Web servers that might want to gain unauthorized access to cookies. The cross-domain cookies we propose use the same basic threat model. In addition, secure XDC channels promote secure and confidential exchange of XDC cookies, mitigate against DNS spoofing attacks, and provide an extra layer of protection against Web site impersonation.

7.4 User control and privacy

Ultimately, the user must be in control of cookie sharing performed by his browser. The comment attributes in both XDC cookies and XDC authorizations should help him in making informed decisions about it.

In addition, the tracking protection framework ('Do Not Track', or *DNT*) nearing completion in the World Wide Web Consortium [5] can be adapted to cover XDC cookies as well. The DNT framework defines the users' rights vis-à-vis tracking by Web sites, the practices required of them to comply with the user preferences, and the technical means to express these preferences and compliance with them. We believe that cross-domain cookies proposed in this paper do not introduce any new concerns that don't already exist for traditional first-party and third-party cookies. To the degree that the DNT framework addresses those concerns, it should address them for XDC cookies as well. Specific compliance rules and technical mechanisms will need to be modified to incorporate a new scope, namely, an XDC channel. (At present, the DNT framework only considers sites and resources within those sites).

Even traditional cookies remain somewhat of a mystery to many end users, but at least they contain the Domain attribute that hints at the cookies' scope and applicability. Names of XDC cookies, on the other hand, are digests of public keys, and do not contain any information that may be recognized by the users. To mitigate this we suggest that browsers maintain a running log of recent use of all persistent XDC cookies capturing the channel name, the host name of the Web server reading or writing the cookie, the date and time of its last access, the type of access, and, possibly, the value set. This log can be used by administrators and advanced users to analyze XDC access patterns and modify their browsers' cookie acceptance rules if needed.

7.5 Compatibility with existing infrastructure

Although all functionality available through traditional cookies can be implemented with XDC cookies, we do not propose phasing them out, even in the long term. Traditional cookies enjoy widespread acceptance and have almost no operational and communication overhead. The two types of cookies should be able to coexist in the same protocol. Web browsers should treat these cookies as completely disjoint: a traditional cookie named X and an XDC cookie named X represent unrelated data even when they are received from (or need to be sent to) the same server. Such clean separation makes XDC cookies simpler to implement, and should ease their adoption by browser manufacturers.

Our proposal makes changes to the HTTP protocol. Any changes to HTTP must interoperate with the existing Web infrastructure. We need to evaluate how XDC-aware actors (i.e., servers, clients, and proxies) interact with those that are not XDC-aware. Three use cases may be considered:

- *XDC-aware client/unaware server.* This is the simplest case: the browser may issue XDC preflight requests, they will fail or return no information. The server will ignore any XDC cookies sent by the browser (if it manages to discover XDC authorizations for the server by other means).
- *XDC-aware server/unaware client.* The server's application must be coded defensively and have a backup implementation that doesn't rely on XDC cookies. The server may test the client by sending the status code `XDC_RESPONSE_REQUESTED` and checking if it receives a repeat request with an `Xdc-Response` header set to `true`.
- *XDC-unaware proxy.* Since preflight authorizations use the OPTIONS method (Section 4.2), and responses to OPTIONS requests are not cacheable [40], XDC-unaware proxies should not be disruptive for this part of our solution.

`XDC_RESPONSE_REQUESTED` responses which initiate repeat requests must set the `Cache-Control` header to `no-cache` to prevent their caching by any proxies. To forestall caching of XDC-specific headers responses to user requests must set the `Cache-Control` header to `no-cache= Xdc-Authorization; no-cache=Xdc-Channel` [40].

8 Conclusion

Cookies provide a simple state management mechanism for HTTP. As currently implemented, they can be shared only between hosts in the same DNS domain (with some limitations). In many cases, however, this is too restrictive, and the ability to share cookies across domains may be required. Although there are technical means to work around the current limitations, they are difficult to implement, costly and sometimes unsafe. Conversely, the same origin policy currently in use on the Web may be too permissive in some cases; it could benefit from a fine-grained access control mechanism if one was developed to support cookie sharing across domain boundaries.

In this paper we introduced a simple authorization model for sharing cookies between disparate DNS domains. Such cookies are written to or read from cross-domain channels (XDC). Both writers and readers are issued XDC authorizations granting appropriate permissions to their holders and binding these permissions cryptographically to the XDC channels' owners. XDC authorizations may be delivered in the HTTP stream that carries XDC cookies themselves, or looked up in the DNS. The binding of an XDC authorization to the host presenting it relies on the trustworthiness of the name resolution process and, therefore, may be vulnerable to pharming and other attacks against the DNS. Secure XDC channels allow their owners to indicate that cookies may be shared only across SSL connections; this mitigates against DNS spoofing and ensures security and confidentiality of the XDC cookies in transit.

Similar to CORS, the Cross-Origin Resource Sharing mechanism implemented in many browsers [22], our solution uses client-cacheable preflight authorizations which should minimize repeat requests and other XDC-related communication overhead. Preflight requests provide the browser with the Web sites' XDC authorizations, and also give additional instructions about XDC cookie handling (such as the frequency of DNS lookups). Since any given Web site is expected to use only a small number of cross-domain channels, XDC authorizations are fairly small (about 1.5 K), and XDC cookies themselves are only marginally bigger than traditional cookies, the overall solution is lightweight. At a modest cost our solution provides a simple and secure mechanism for cross-domain cookie sharing on the Web.

Endnotes

[a]Another HTTP state management standard has recently been proposed (RFC 6265) [41]. It obsoletes RFC 2965 and augments RFC 2109.

[b]In this paper we do not consider IP addresses used in HTTP URLs (and cookies' Domain attribute). Their direct use is generally discouraged [42].

[c]RFC 2965 also defines a new header, `Set-Cookie2` [3]. The differences with the older header are slight, and we will not discuss it further.

[d]Note that, like CORS, we make preflight requests to collect authorization information prior to fulfilling user requests.

[e]DNS resource records used to store arbitrary text [43]

[f]A newer version of our prototype can generate XDC authorizations using the ECDSA algorithm. With 192-bit elliptic curves, which provide security comparable to 1024-bit RSA keys [44], fully-encoded authorizations are about 200 bytes shorter.

[g]For example, the one used by the browsers' SSL/TLS implementations.

Competing interests
The author declares that he has no competing interests.

Received: 5 February 2013 Accepted: 5 February 2013
Published: 11 April 2013

References

1. Kristol DM (2001) HTTP Cookies: Standards, privacy, and politics. ACM Trans Internet Technol 1(2): 151–198
2. Kristol D, Montulli L (1997) HTTP State Management Mechanism. IETF, RFC 2109
3. Kristol D, Montulli L (2000) HTTP state management mechanism. IETF, RFC 2965
4. HTTPOnly (2007) Open Web Application Security Project (OWASP)
5. Tracking Protection Working Group. (http://www.w3.org/2011/tracking-protection/). W3 Consortium (2012)
6. Zalewski M (2009) Browser security handbook. Google, Inc
7. Persistent Client State HTTP Cookies. Netscape communications corporation (undated)
8. How Businesses are Using Web 2.0: A McKinsey global survey. McKinsey and Company (2007)
9. O'Reilly T (2005) What Is Web 2.0: Design patterns and business models for the next generation of software. O'Reilly Network
10. Phifer G (2011) Hype cycle for web and user interaction technologies. Gartner, Inc
11. Hughes J, Cantor S, Hodges J, Hirsch F, Mishra P, Philpott R, Maler E (eds) (2005) Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0
12. Lockhart H, Campbell B (eds) (2008) Identity provider discovery service protocol and profile
13. Jang D, Venkataraman A, Sawka GM, Shacham H (2011) Analyzing the crossdomain policies of flash applications. In: Proc. of the Web 2.0 Security and Privacy Workshop
14. Kontaxis G, Antoniades D, Polakis I, Markatos EP (2011) An Empirical study on the security of cross-domain policies in rich internet applications. In: Proc. of the 4th European Workshop on System Security
15. Pettersen Y (2008) HTTP state management mechanism v2. IETF. Internet Draft draft-pettersen-cookie-v2-05
16. Callaghan PJ, Howland MJ, Pritko SM (2008) Method, system and program products for sharing state information across Domains. U.S. Patent and Trademark Office. Patent Application Publication US 2008/0027824 A1

17. Guo R, Zhou B (2008) Cross Cookie: A cookie protocol for web mashups. In: Proc. of the 2008 International Symposium on Electronic Commerce and Security, 416–420
18. Hickson I (ed) (2011) Web Storage. W3 Consortium, W3C Candidate Recommendation 08/12/2011
19. Hickson I (ed) (2011) HTML5 Web Messaging. W3 Consortium, W3C Working Draft 10/20/2011
20. window.postMessage (https://developer.mozilla.org/en-US/docs/DOM/window.postMessage). Mozilla Developer Network (2012)
21. Hanna S, Shin R, Akhawe D, Boehm A, Saxena P, Song D (2010) The Emperor's New APIs: On the (In) secure usage of new client-side primitives. In: Proc. of the 4th Web 2.0 Security and Privacy Workshop
22. van Kesteren A (ed) (2010) Cross-origin resource sharing. World Wide Web Consortium, W3C Working Draft 07/27/2010
23. Zakas N (2010) Cross-domain Ajax with cross-origin resource sharing. NCZOnline, 2010
24. Farrell S, Housley R (2002) An internet attribute certificate profile for authorization. IETF, RFC 3281
25. Karlof CK, Shankar U, Tygar D, Wagner D (2007) Locked cookies: Web authentication security against Phishing, Pharming, and active attacks. University of California at Berkeley, Technical Report UCB/EECS-2007-25
26. Information Technology - Open Systems Interconnection - The Directory: Authentication Framework. ITU-T Recommendation X. 509 (1997)
27. Cooper D, Santesson S, Farrell S, Boeyen S, Housley R, Polk W (2008) Internet X.509 public key infrastructure certificate and Certificate Revocation List (CRL) profile. IETF, RFC 5280
28. Mockapetris P (1987) Domain names – Implementation and specification. IETF, RFC 1035
29. Eastlake DE (1999) The kitchen sink DNS resource record. IETF, Internet Draft draft-ietf-dnsind-kitchen-sink-02
30. The Legion of the Bouncy Castle: Welcome (http://www.bouncycastle.org/java.html). The Legion of the Bouncy Castle (2011)
31. Apache Tomcat, http://tomcat.apache.org/. Apache Software Foundation (2011)
32. Apache Module mod_rewrite http://httpd.apache.org/docs/current/mod/mod_rewrite.html. Apache Software Foundation (2011)
33. Apache Module mod_headers http://httpd.apache.org/docs/current/mod/mod_headers.html. Apache Software Foundation (2011)
34. ISC BIND Nameserver - Howtos, Links, Whitepapers. (http://www.bind9.net/). BIND9.NET/BIND9.ORG (2010)
35. Extensions https://developer.mozilla.org/en/Extensions. Mozilla developer network (2011)
36. Rabinovich P (2011) Cross-domain cookies, https://sourceforge.net/projects/xdccookies/. SourceForge.net
37. Tappenden AF, Miller J (2009) Cookies: A deployment study and the testing implications. ACM Trans Web 3(3): 1–49
38. Schneider F, Agarwal S, Alpcan T, Feldmann A (2008) The new web: characterizing AJAX traffic. In: Proc. of the 9th International Conference on Passive and Active Network Measurement, 31–40
39. Secure Hash Signature Standard. FIPS Publication 180-2 (2002)
40. Fielding R, Gettys J, Mogul J, Frystyk H, Masinter L, Leach P, Berners-Lee T (1999). Hypertext Transfer Protocol – HTTP/1.1. IETF, RFC 2616
41. Barth A (2011) HTTP state management mechanism. IETF, RFC 6265
42. A guide to building secure web applications and web services. Open Web Application Security Project (OWASP) (2005)
43. Rosenblum R (1987) Using the domain name system to store arbitrary string attributes. IETF, RFC 1464
44. Barker E, Barker W, Burr W, Polk W, Smid M (2012) Recommendation for key management – Part 1: General. NIST Special Publication Revision 3: 800–857

doi:10.1186/1869-0238-4-13
Cite this article as: Rabinovich: Secure cross-domain cookies for HTTP. *Journal of Internet Services and Applications* 2013 **4**:13.