Network Working Group
Internet-Draft
Updates: 2616 (if approved)
Intended status: Standards Track
Expires: September 7, 2012

Y. Pettersen
Opera Software ASA

March 6, 2012

A context mechanism for controlling caching of HTTP responses
draft-pettersen-cache-context-06

Abstract

A common problem for sensitive web services is informing the client, in a reliable fashion, when a password protected resource is no longer valid because the user is logged out of the service. This is, in particular, considered a potential security problem by some sensitive services, such as online banking, when the user navigates the client's history list, which is supposed to display the resource as it was when it was loaded, not as it is the time the user navigates to it.

This document presents a method for collecting such sensitive resources into a group, called a "Cache Context", which permits the server to invalidate all the resources belonging in the group either by direct action, or according to some expiration policy. The context can be configured to invalidate not just the resources, but also specific cookies, HTTP authentication credentials and HTTP over TLS session information.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference

material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 7, 2012.

Copyright Notice

Table of Contents

1. Introduction

An early problem seen with HTTP was that, since it is inherently stateless, there is no direct way to tell that two separate requests are related, in particular that the requests originated from the same user. While it is possible to encode the information in URLs or queries, these methods are difficult to secure and it is also difficult to maintain that information.

Finally, much of the problem was solved by the introduction of HTTP Cookies [RFC6265] and the HTTP Authentication methods [RFC2617].

One problem remains, though: History list navigation and access to temporarily cached resources in web applications handling sensitive data, such as online banking. The user may have logged out of the service, or have been logged out automatically, and it is therefore difficult for the client to tell whether to display a given resource, not display it, or display it only after revalidating.

A typical use case will be that Alice goes to her online banking site, checks her accounts, performs a couple of transactions and, then, logs out of the bank. The bank might want to permit Alice to navigate in her browser's history session while she is logged in, for example, to recheck how much money is available in her account, but prevent her from seeing any such information after she has logged out. The reason is that, if she forgets to close her session (after logging out) and then leaves her computer for a minute, Eve might sneak over to peek at her colleague's account information by navigating the history list in Alice's client.

Some providers of sensitive web services, for example banks, consider failure to revalidate when displaying a cached document, or during history navigation, a security problem. Some of these providers have put great efforts into making sure the client is always revalidating a page before displaying it, even while navigating history (which is quite the opposite of what [RFC2616] sec. 13.13 recommends). This has led to numerous strategies, such as the use of scripting and using the "must-revalidate" Cache-Control directive as a revalidate indication for history navigation.

Use of most of these methods not only results in more traffic to the websites, but may also reduce the usability of the service. For example, such methods may prevent a user from reviewing, or correcting, data entered or presented earlier in the process, or to print or save data presented earlier (for example, a receipt), because the document is re-rendered differently.

While it is possible to use cookies and credential information to

group such documents, these methods are not fine-grained enough to distinguish the sensitive parts of a site from the non-sensitive parts. That kind of information is only available to the service manager, and potentially the user, but not to the client.

This document presents a mechanism that permits a webservice to group webpages by associating them in a named context, a "Cache Context", that can have a specific expiration time or be specifically discarded by the service. When a context is expired, or discarded, this expires all documents that the client has that are part of the context, meaning that they should either be deleted from the client's cache or revalidated before being displayed to the user.

The mechanism is implemented as an extension to the HTTP Cache-Control Header, but will also require access to state information about HTTP Cookies and various forms of authentication credentials, such as HTTP Authentication credentials and SSL Client Certificate session states.

1.1. Terminology

HTTP Resource: A resource loaded from a HTTP [RFC2616] or HTTP over TLS (HTTPS) [RFC2818] server

2. How Cache Contexts work

2.1. What is a Cache Context?

A Cache Context is a group of HTTP resources served from a specific HTTP server, or group of HTTP servers, that are associated with a name unique on that server, or within that group of servers. The same name used from a different server, or a server that is not part of the group, becomes a different context. Any given server can use multiple contexts.

An HTTP resource from a server for which a context is defined does not become part of the context unless the server explicitly informs the client that it is a member of that context.

Such a group of resources may, for example, consist of all the sensitive and password protected pages in a netbank web application, but none of the images used when displaying the information. A context may also just contain a group of temporary documents that are not intended to be persistently cached.

A Cache Context can be created with policies that define when it is to be discarded, and what actions it should then take, aside from

invalidating all the resources that are part of the context. Such extra actions may include deleting or invalidating cookies, HTTP authentication credentials or SSL session information.

2.2. Life of a Cache Context

The Cache Context enters existence the first time the server sends a directive that informs the client about the context, and what boundaries the context has. Resources are added to the context each time the server sends a new resource with a directive specifying that it is part of the context.

The server (or servers) that can serve resources in a context are either the server defining the context (this is the default), or the initial server can specify that the context spans the same group of servers that will receive a specific HTTP cookie [RFC6265]. HTTP cookies can be set by servers to only be returned by the client to the server that sent them, or to be sent to all servers within the domain which the sending server belongs to (there are some security related restrictions to how extensive such domains can be).

For example, when Alice logs into her online bank, the server tells her client that the "onlinebanking" context has been created, and, for each successive action she makes to display her account statements and filling in forms with payment information, the server informs the client that the resulting pages are also part of the "onlinebanking" context.

This continues until the context is discarded, an event that can come about as a result of several conditions:

- Alice logs out of her online bank, and the server sends a "Discard Context" notice to the client.
- The server has specified how long the context may live, and, when the limit expires, the context is automatically discarded.
- The server can also tie the context to an HTTP State Management cookie[RFC6265], and when this cookie expires, the context is also automatically discarded.
- The client SHOULD offer the user the possibility to discard a context in the form of a logout button.
- The client is shut down.

2.3. What happens when a context is discarded?

When a context is discarded, the resources that are part of the context are deleted or marked for deletion (e.g., if they cannot be deleted immediately due to internal book-keeping). When a resource is marked for deletion, its status in the client's cache will be as if it had been received with max-age set to 0 (zero); that is, it has expired and cannot be displayed to the user unless its freshness has been confirmed by the server after a validation request. This also applies to such documents that would be displayed after the user navigated in the browser's history. Preferably, the resource SHOULD NOT be used if the client requests the same URI as that of the resource, even conditionally, after the context has been discarded, except during history list navigation.

If the context is associated with an HTTP cookie, discarding the context also causes the client to automatically discard the cookie.

The server can also specify that when a context is discarded, stored credentials that are valid for the server(s) that the context spans should also be deleted. Such credentials should include, but should not be limited to, HTTP Authentication credentials and SSL Session information. Credentials stored in a client's password management utility MUST NOT be discarded during this process.

Resources gathered in a context MUST NOT be kept persistently cached by the client, but MUST be discarded when it shuts down, even if the context is still valid. Please note that this does not prevent a client from storing the resource on a disk drive for the duration. If a server wants to prevent a resource from being stored on a disk drive it MUST indicate this request with the "no-store" Cache-Control directive.

A resource served with an explicit discard context instruction from the server is not part of the context and MUST NOT be invalidated, and the server SHOULD NOT include sensitive information in such resources.

2.4. Server role

A service that wants to use Cache Contexts must update the HTTP Cache-Control headers sent by each of the resources it wants to make part of the context. Only the entry point(s) into the service will need to send a directive with expiration details and other advanced information, all other resources need only send a directive specifying the name of the context.

When a service has exit points, such as a logout button, these

special resources should send a special directive informing supporting clients that they must discard the context.

A server MAY send updated Cache Context directives to the client, and need not resend unchanged attributes in such updates.

2.5. Client role

The client will parse the Cache-Control headers, and when it recognizes the Cache Context directives it will take one of several actions:

- Create the context as defined, if it does not already exist.
- If the server sends updated attributes for the context, update the relevant attributes without changing other attributes.
- Add the resource to the context, unless the context is being discarded as a result of the current directive.
- If the directive specifies that the context must be discarded immediately, the client MUST proceed to invalidate all the resources contained in the context, as specified above.

The client MUST maintain a list of active contexts for each server or group of servers, their policies and which resources are associated with them.

A server MAY specify that a resource belongs in two different contexts by sending two directives in the same response. In such cases the resource MUST be invalidated when the first context is discarded.

2.6. Effects on clients that do not support Cache Contexts

All the information about the Cache Contexts and actions on them are contained in new HTTP Cache-Control directives. Clients that do not recognize these directives will ignore them. They will follow the Cache-Control directives they recognize, and otherwise act as they would if the Cache Contexts did not exist.

3. Context Specification Syntax

3.1. ABNF syntax

This ABNF has the same syntax as is used in [RFC2616], with the addition of Incremental Alternatives from [RFC5234] section 3.3.

This syntax expands the cache-response-directive part of the Cache-Control header ABNF in RFC 2616 sec. 14.9.

```
cache-response-directives =/
       "context" "=" context-name *[ ";" context-attributes]
     | "discard-context" "=" context-name *[; context-name]

context-name = token

context-attributes =
       "max-age" ":" delta-seconds
     | "max-idle" ":" delta-seconds
     | "cookie" ":" token
     | "authenticated"
     | "include-credentials"
     | "no-persistent-history"
     | extension-attributes

extension-attributes = token [":" token]

delta-seconds =  1*DIGIT
```

3.2. Context directive

The Context directive MUST be included with every response that is included in a context. It always starts with a context-name, which MUST be unique within the given service.

The context-name, which is case-insensitive, is the only required field in the directive. All other attributes are optional, and need only be specified when the attribute is updated.

By default, a given context on one server is not associated with a context by the same name for another server, but if a group of servers share a common HTTP cookie, they can participate in a common context by using the same name.

Additionally, the directive may contain one or more context-attributes that specify how the context as well as additional information should be handled by the client. These attributes need not be sent with every response, but when an attribute is sent with a later response, it will update the context with the new information, and any previous information for that attribute will be overwritten.

max-age This attribute indicates how many seconds the context should be kept alive. A value of zero (0) means that the context should be discarded immediately. A server may send this directive to extend the lifetime of the context, e.g when the context nears the end of its lifetime. If this attribute is not defined, the context may live until the client is shut down, or a discard event is received by the client.

max-idle This attribute indicates how many seconds the context should be kept alive after the last time a resource in the context was accessed. If this attribute is not defined, or the value is zero (0), the context may live until the client is shut down, or a discard event is received by the client.

cookie This attribute names the most specific HTTP State Management Cookie [RFC6265] visible to the resource sending when this attribute is received by the client. Cookies received and accepted in the current response MAY be considered as part of evaluating this attribute. The named cookie is used for three purposes. First, its expiration date also becomes one of the expiration dates of the context (an earlier max-age or max-idle takes precedence), and if the server deletes the cookie, it automatically deletes the context as well. Second, if a cookie is valid across multiple hosts (that is, a domain), the given context is also valid across the same hosts. Third, if the context expires or is discarded before the cookie expires, the named cookie MUST also be deleted when the context is discarded. Multiple cookies may be defined by the server. By default, the context is not assiciated with a cookie.

authenticated This attribute specifies that the context shall be valid across the domain of the HTTP authentication credential currently valid for the resource. The domain MUST at least include the entire server, but multiple hosts may be included if it is supported by the authentication mechanism, as for example Digest Authentication [RFC2617] do. When the context is discarded, the authentication credential is also discarded. If the credential is invalidated or destroyed, the context must also be discarded. By default, the context is not associated with an authentication credential.

include-credentials This attribute informs the client that when this context is discarded it MUST also destroy all HTTP Authorization credentials, SSL/TLS Client Certificate authenticated sessions, and other credentials that are valid for the resources that are part of the context, thus logging the user out of the service. By default, credentials are not included when discarding a context.

no-persistent-history This attribute informs the client that it SHOULD NOT remember resources that are part of this context as part of a persistent history mechanism. That is, the client SHOULD maintain a history list in the document window or "tab" viewed by the user (as described in [RFC2616] Section 13.13) , but it SHOULD NOT keep any information about the URLs visited for a persistent browsing history. By default, the client MAY remember the used resources as part of a persistent history mechanism.

delta-seconds All periods used in the Cache Context attributes are measured in seconds.

3.3. Discard-Context directive

A server can send this directive when it wants the client to immediately discard the named context(s) (which includes the extra actions specified in Section 2.3 when specified for the context).

Multiple context names separated by ";" can be specified in a single directive, or multiple directives can be used.

3.4. Extensions

Extension attribute names SHOULD be documented in an RFC.

4. Examples

4.1. No expiration

```
Request-URI: http://www.example.com/initial_page

Response:
Cache-Control: context=simplecontext

Request-URI: http://www.example.com/page2

Response:
Cache-Control: context=simplecontext

Request-URI: http://www.example.com/page3

Response:
Cache-Control: context=simplecontext

Request-URI: http://www.example.com/page4

Response:
Cache-Control: max-age=3600

Request-URI: http://www.example.com/final_page

Response:
Cache-Control: discard-context=simplecontext
```

This example loads 3 resources, "initial_page", "page2" and "page3", as part of the Cache Context "simplecontext". By default, this context lives until the client is shut down. In this case, however, the context is discarded by the response to the request for "final_page". After the context has been discarded, all future attempts to view "initial_page", "page2" or "page3" will result in an "If-Modified-Since" validation request to the server, or a completely new request, because the responses are no longer valid.

"Page4" is not part of the context, and is not discarded by the "final_page" action, and no "If-Modified-Since" request will be sent for this resource until 3600 seconds (1 hour) after it was originally loaded.

4.2. With expiration

```
Request-URI: http://www.example.com/initial_page

Response:
Cache-Control: context=expirecontext;max-age=900



Request-URI: http://www.example.com/page2

Response:
Cache-Control: context=expirecontext



Request-URI: http://www.example.com/page3

Response:
Cache-Control: context=expirecontext
```

This example loads 3 resources, "initial_page", "page2" and "page3", as part of the Cache Context "expirecontext". By default, this context lives for at most 15 minutes (900 seconds). After the context has been discarded (in this case, after 15 minutes), all future attempts to view "initial_page", "page2" or "page3" will result in an "If-Modified-Since" validation request to the server, or a completely new request, because the responses are no longer valid.

4.3. With server-only cookie

```
   Defined cookie (server only):
      ExampleSession=UserId; max-age=900; domain=www.example.com


   Request-URI: http://www.example.com/initial_page

   Response:
   Cache-Control: context=cookiecontext;cookie=ExampleSession



   Request-URI: http://www.example.com/page2

   Response:
   Cache-Control: context=cookiecontext


   Request-URI: http://www.example.com/page3

   Response:
   Cache-Control: context=cookiecontext



   Request-URI: http://www.example.com/final_page

   Response:
   Cache-Control: discard-context=cookiecontext
```

This example loads 3 resources, "initial_page", "page2" and "page3", as part of the Cache Context "cookiecontext" which is associated with the cookie "ExampleSession". By default, this context lives until the cookie expires 15 minutes (900 seconds) after it was set. In this case, however, the context is discarded by the response to the request for "final_page". After the context has been discarded, the cookie "ExampleSession" no longer exists even if it was not yet expired, and all future attempts to view "initial_page", "page2" or "page3" will result in an "If-Modified-Since" validation request to the server, or a completely new request, because the responses are no longer valid.

4.4. With cookiedomain

```
Defined cookie:
   ExampleDomainSession=UserId; max-age=900; domain=.example.com

Request-URI: http://www.example.com/initial_page

Response:
Cache-Control: context=domaincontext;cookie=ExampleSession


Request-URI: http://server2.example.com/page2

Response:
Cache-Control: context=domaincontext


Request-URI: http://server3.example.com/page3

Response:
Cache-Control: context=domaincontext

Request-URI: http://final.example.com/final_page

Response:
Cache-Control: discard-context=domaincontext
```

This example loads 3 resources, "initial_page", "page2" and "page3" from different servers, as part of the Cache Context "domaincontext" which is associated with the cookie "ExampleDomainSession" which is sent to the entire example.com domain. This causes "domaincontext" to apply to all servers in the example.com domain, too. By default, this context lives until the cookie expires 15 minutes (900 seconds) after it was set. In this case, however, the context is discarded by the response to the request for "final_page". After the context has been discarded, the cookie "ExampleDomainSession" no longer exists even if it was not yet expired, and all future attempts to view "initial_page", "page2" or "page3" will result in an "If-Modified-Since" validation request to the server, or a completely new request, because the responses are no longer valid.

5. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

6. Security Considerations

If two independent web applications that use the same name for their contexts are hosted on the same server, or within the domain covered by one or both of the contexts, they are likely to interfere with each other. This can happen if the user uses both applications while both contexts are valid, possibly causing some loss of functionality and information if a context is discarded, or prolonged exposure of information if the session is extended. Such interference can only be avoided by choosing context names that are not shared among independent web applications.

When using the Cookie attribute, which expands the context to a cookie's domain, this specification relies on the same security safeguards that are used by the client when accepting the cookie in order to avoid interfering with web applications in other services that are using the same context name. Given the wide variety of domain name hierarchies used by TLD administrators, it is presently possible, unless prevented by client specific heuristics, for a server to share a cookie with all servers within a registry-like part of a TLD, such as the "co.uk" Domain Name hierarchy. This kind of interference may also occur within smaller domain name segments.

A possible method to avoid or limit such interference could be to require clients to perform an evaluation of the context directive's cookie specification from the resource's environment, which might associate it with a different cookie. Such an evaluation would most likely result in undesirable processing overhead, and is therefore not included in this specification.

7. References

7.1. Informative References

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

7.2. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP

Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

Appendix A. Open Issues

Should the client indicate support for Cache Contexts? Is it necessary to do so? If so, how should support be indicated? A possiblity is an HTTP header with a directive indicating support.

Should the client indicate that it has accepted a particular context and is using it? If so, how should it indicate it? Possible solution: The above mentioned header directive could contain a list of active contexts.

Should the client, when automatically discarding a context, replace the viewed document with a "you have been logged out of the service" document? Or should the last viewed page continue to be displayed? If the document is replaced, how should this situation be handled when a server specifies an unreasonably short expiration time?

How should a client interpret non-context Cache-Control directives in the same header? Given that such directives are likely intended to place more restrictive non-context expiration policies on the resource than is necessary for clients that do support Cache Contexts, the best solution may be that clients supporting Cache Contexts should ignore at least the "no-cache", "max-age=0" and "must-revalidate" directives for resources that are part of a context, all of which are implied when the cache-context is discarded.

How should responses to requests using methods that have side effects, such as POST, be handled after a context has been discarded? Such responses should most likely not be revalidated automatically. The best option may be to require the client to replace the resource with an information message about the resource not being available anymore.

Author's Address

Yngve N. Pettersen
Opera Software ASA
Waldemar Thranes gate 98
N-0175 OSLO,
Norway

Email: yngve@opera.com