RFC 6265 : HTTP State Management Mechanism

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 29 avril 2011. Dernière mise à jour le 30 avril 2011

Date de publication du RFC : Avril 2011

http://www.bortzmeyer.org/6265.html

Les "*cookies*", ces célèbres gâteaux du Web ont déjà fait l'objet d'innombrables articles, par exemple pour critiquer les conséquences pour la vie privée d'un mécanisme qui permet de suivre l'utilisateur à la trace (voir par exemple l'introduction de la CNIL <http://www.cnil.fr/vos-libertes/vos-traces/les-cookies/>). Mais on connait moins leur spécification technique, ou plutôt le fait que, mal spécifiés dès le début, ces gâteaux ont été mis en œuvre de diverses manières et qu'il est très difficile de décrire intégralement leur fonctionnement. C'est la tâche de notre RFC 6265 [1], d'être, normalement, la norme technique complète et faisant autorité sur les "*cookies*", en capturant l'ensemble des points techniques sur les gâteaux en une seule norme. Les textes précédents étaient la description originale de Netscape <http://web.archive.org/web/20020803110822/http://wp.netscape.com/newsref/std/cookie_spec.html> (on notera avec ironie que cette spécification privée n'est plus accessible que via WebArchive), puis le RFC 2109, puis le RFC 2965, que notre RFC remplace et annule. Les documents précédents étaient loin de spécifier tous les aspects des petits gâteaux.

Le point de départ est que le protocole HTTP, le plus utilisé du Web est **sans état**. Lors d'une visite à un site Web, l'utilisateur clique successivement sur plusieurs liens, visite plusieurs pages, et chaque téléchargement d'une page est une requête HTTP séparée des autres, sans rien qui permette de les corréler (par exemple pour se souvenir des préférences exprimées par un utilisateur lors de l'arrivée sur la première page). Les gâteaux visent à combler ce manque. La norme définit deux en-têtes HTTP, `Set-Cookie` qui permet au **serveur** HTTP de déposer un gâteau sur l'ordinateur de l'utilisateur et `Cookie` qui permet au **client** HTTP (typiquement le navigateur) de renvoyer un gâteau préalablement déposé. Lorsque le serveur reçoit le gâteau, il le compare à ceux qu'il a envoyés et reconnait ainsi un utilisateur déjà vu. HTTP devient alors un protocole **avec état**.

Ce n'est pas que les gâteaux soient une solution géniale : imposés par un acteur du Web particulier, Netscape, ils n'avaient pas fait l'objet d'un examen sérieux par une SDO et ont des tas d'inconvénients

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc6265.txt`

techniques et politiques. Néanmoins, comme ils sont largement utilisés dans le Web, il était préférable qu'une documentation sérieuse de leur syntaxe et de leur comportement existe, et c'est le rôle de ce RFC 6265.

La section 1 de ce RFC résume le principe des gâteaux. Un gâteau inclus un certain nombre de métadonnées qui permettent au client HTTP de savoir si et quand il doit retransmettre le "*cookie*". Par exemple, on peut spécifier la période pendant laquelle le gâteau reste valable, les serveurs à qui on doit le renvoyer, etc. Du fait de leur spécification hâtive et non collective, les gâteaux ont bien des faiblesses techniques. Par exemple, l'attribut `secure` dans les métadonnées ne fournit aucune garantie d'intégrité en dépit de ce que son nom pourrait faire croire. D'autre part, les gâteaux ne sont pas spécifiques à un port donné donc, si plusieurs serveurs tournent sur des ports différents de la même machine, ils peuvent se voler leurs gâteaux mutuellement. Mais, d'une manière générale, ce RFC n'essaie pas de réparer la situation, mais de documenter les gâteaux tels qu'ils sont réellement produits et consommés sur le Web.

Bien, assez râlé, qui doit lire ce RFC ? Il y a deux audiences, les développeurs d'applications côté serveurs (ou, plus exactement, des bibliothèques qui sont utilisées par ces applications) qui produisent et consomment des gâteaux, et les développeurs des logiciels clients, notamment les navigateurs Web. Pour maximiser les chances d'interopérabilité, les serveurs devraient se montrer très prudents et ne produire que des gâteaux parfaitement conformes à la norme (section 4 du RFC, qui décrit le profil restreint), alors que les clients doivent être plus tolérants et accepter la grande variété des gâteaux actuellement fabriqués (section 5, sur le profil libéral), car tous les serveurs ne suivent pas le profil restreint.

Avant de lire le gros du RFC, un petit détour par la norme HTTP, le RFC 7230, peut être utile, puisque notre RFC en reprend le vocabulaire.

La section 3 décrit les généralités sur les gâteaux. Ceux-si sont envoyés par un en-tête `Set-Cookie` dans la **réponse** HTTP, puis retournés dans un en-tête `Cookie` dans la **requête** suivante. L'état est donc stocké chez le client mais, en pratique, la taille des gâteaux étant limité, le serveur stocke l'essentiel de l'état et le gâteau ne sert que d'index pour retrouver une session particulière. Un serveur peut utiliser `Set-Cookie` dans n'importe quelle réponse (y compris les erreurs comme 404) et le client doit le gérer (sauf pour les réponses de type 1xx, qui sont seulement « pour information »). Plusieurs gâteaux peuvent être envoyés, chacun dans un en-tête `Set-Cookie` différent (le client, lui, peut utiliser un seul en-tête `Cookie` pour envoyer plusieurs gâteaux). Un exemple classique est l'envoi d'un identificateur de session (ici, SID, pour "*Session ID*"). Le serveur envoie :

```
Set-Cookie: SID=31d4d96e407aad42
```

et à la requête suivante, le client renverra :

```
Cookie: SID=31d4d96e407aad42
```

Le serveur peut réduire la portée du gâteau avec les attributs `Path` et `Domain`. Ici, par exemple, le gâteau doit être renvoyé pour tous les URL dans `example.com` :

```
Set-Cookie: SID=31d4d96e407aad42; Path=/; Domain=example.com
```

Ici, un serveur prudent utilise en plus les attributs `Secure` et `HttpOnly`. En outre, il envoie deux gâteaux, SID et lang (chacun étant un doublet nom-valeur) :

```
Set-Cookie: SID=31d4d96e407aad42; Path=/; Secure; HttpOnly
Set-Cookie: lang=en-US; Path=/; Domain=example.com
```

Le client peut les renvoyer en un seul en-tête :

```
Cookie: SID=31d4d96e407aad42; lang=en-US
```

Ici, le serveur fixe une date d'expiration au gâteau :

```
Set-Cookie: lang=en-US; Expires=Wed, 09 Jun 2021 10:18:14 GMT
```

Après ces exemples tirés du RFC, voici maintenant un exemple réel entre un site Web écrit en PHP et un navigateur. Le trafic a été capturé et formaté avec tshark :

```
# Le serveur envoie :
Set-Cookie: PHPSESSID=euk3daoldp4s3j90uocetas3a0; path=/\r\n

# Le client répond (d'autres gâteaux avaient été transmis) :
Cookie: __utma=85646793.1868252067.1190397623.1190397623.1190397623.1; \
     PHPSESSID=euk3daoldp4s3j90uocetas3a0; \
     culture=fr\r\n
```

Autre façon de voir les gâteaux, se servir du client en ligne de commande curl :

```
% curl -v http://twitter.com/ > /dev/null
...
< Set-Cookie: guest_id=130393513906128799; path=/; expires=Fri, 27 May 2011 20:12:19 GMT
< Set-Cookie: _twitter_sess=BAh7CDoPY3JlYXRlZF9hdGwrCPc8l5gvAToHaWQiJWY3NTg3YTU1YTgwOWE2%250AODcxMTA5M2NiNzNiOWM
```

curl permet même de renvoyer ces gâteaux au serveur :

```
% curl --cookie _twitter_sess=BAh7CDoPY3JlYXRlZF9hdGwrCPc8l5gvAToHaWQiJWY3NTg3YTU1YTgwOWE2%250AODcxMTA5M2NiNzNiO
...
> Cookie: _twitter_sess=BAh7CDoPY3JlYXRlZF9hdGwrCPc8l5gvAToHaWQiJWY3NTg3YTU1YTgwOWE2%250AODcxMTA5M2NiNzNiOWMxODR
...
```

La section 4 définit rigoureusement les obligations d'un serveur sérieux, qui ne cherche pas à exploiter les possibilités de la norme à fond et qui veut être sûr que tous les clients pourront traiter correctement ses gâteaux. Sa section 4.1.1 définit la syntaxe formelle à suivre, en ABNF. Les valeurs qui ne sont pas réduites à un strict sous-ensemble de caractères sûrs, pour maximiser les chances d'être transmises sans accident, sont encodées en Base64. Les dates s'expriment avec le format du RFC 1123 (mais certains serveurs non compatibles avec ce profil envoient des dates sur deux chiffres...). Certains clients stockent les dates dans un entier de seulement 32 bits donc il est possible que les choses se passent mal à partir de 2038.

Le serveur peut ajouter des attributs aux gâteaux. Ainsi, l'attribut `Expires` (section 4.1.2.1) permet de spécifier la date limite de consommation du gâteau (« À consommer avant le 29 avril 2011 ») et

l'attribut `Max-Age` indique la durée de vie maximale (le RFC prévient que `Max-Age` est moins souvent mis en œuvre que `Expires` par les clients HTTP). Le gâteau peut durer moins longtemps que cela, par exemple parce que le navigateur Web a décidé de faire de la place, ou bien parce que l'utilisateur l'a détruit explicitement (chose que permettent tous les bons navigateurs, vues les conséquences négatives des gâteaux pour la vie privée).

Important pour la sécurité, l'attribut `Domain` permet d'indiquer le domaine des serveurs à qui le client doit renvoyer le gâteau. Attention, les sous-domaines de ce domaine sont également acceptés. Si le serveur envoie `Domain=example.org` lors du `Set-Cookie`, le client renverra le gâteau à, par exemple, `www.example.org` et `mail.example.org`. (Et si un serveur distrait met un gâteau avec `Domain=com`? Il sera renvoyé à tous les domains en `.com`? Cela dépend du navigateur; beaucoup ignorent cet attribut `Domain` s'il correspond à un domaine d'enregistrement comme `.fr` ou `.co.uk` mais il n'existe pas de moyen officiel d'en connaître la liste donc ces navigateurs dépendent de listes incomplètes comme <`http://publicsuffix.org`>. Voir la section 5.3 pour une discussion complète. À noter que Konqueror utilise une méthode encore pire qui n'a jamais été corrigée <`http://bugs.kde.org/show_bug.cgi?id=163774`>.)

Le serveur peut restreindre encore plus l'utilisation du gâteau qu'il envoie avec l'attribut `Path` (section 4.1.2.4). Celui-ci indique quelle section du site Web a besoin du gâteau. Si on met `Path=/accounting` et que le domaine est `www.example.com`, seuls les URL commençant par `http://www.example.com/accountin` recevront le gâteau.

Enfin, deux autres attributs permettent d'améliorer la (faible) sécurité des gâteaux, `Secure`, qui impose au client de ne renvoyer le gâteau qu'au dessus d'une connexion « sûre » (typiquement HTTPS, pour éviter les attaques par FireSheep) et `HttpOnly` qui restreint le gâteau à ce protocole.

Une fois que le serveur a créé le gâteau et l'a envoyé avec les attributs qu'il juge utile, il doit être prêt à recevoir des gâteaux retransmis par le client. La section 4.2 décrit l'en-tête `Cookie` des requêtes HTTP, qui sert à cela. (Voir les exemples plus haut.)

La section 4 décrivait un profil réduit des gâteaux. Un client HTTP sérieux qui veut pouvoir comprendre tous les serveurs existants dans la nature, y compris ceux qui ne s'en tiennent pas à ce profil réduit, doit, lui, lire la section 5, qui décrit l'intégralité du très compliqué comportement des gâteaux. Cette section reprend le plan de la 4, en détaillant toutes les horreurs qu'on peut rencontrer dans le monde réel.

Par exemple, le format des dates (section 5.1.1), utilisées dans des attributs comme `Expires`, permet à peu près toutes les variations possibles. Écrire un analyseur complet n'est pas trivial, d'autant plus que le RFC ne décrit pas une grammaire indépendante du contexte mais un algorithme à état. (Si un pro de l'analyse syntaxique peut fournir une grammaire plus déclarative de cette syntaxe, je serais intéressé...)

De même, l'analyse du `Set-Cookie` en section 5 est plus complexe car le client doit accepter des `Set-Cookie` plus variés, par exemple avec des espaces en plus.

La section 5.3, qui n'a pas d'équivalent dans la section 4, décrit la façon dont les clients, les navigateurs Web, doivent stocker les gâteaux et les renvoyer. Elle rappelle la liste des attributs à stocker avec chaque gâteau (par exemple, s'il doit être renvoyé uniquement sur une connexion sécurisée ou bien si ce n'est pas nécessaire). Elle est très longue et le pauvre auteur de navigateur Web a donc bien du travail.

En parlant du programmeur, justement, la section 6 décrit les questions que pose l'implémentation de cette spécification. Parmi elles, la question de la taille du magasin où sont stockés les gâteaux. Le

client Web n'est nullement obligé de fournir un magasin de taille infinie. Néanmoins, le RFC demande que le client accepte **au moins** quatre kilo-octets par gâteau, cinquante gâteaux par domaine et trois mille gâteaux en tout (ce qui permet déjà une belle indigestion). Conséquence de cette limite : un serveur doit s'attendre à ce qu'un client ne renvoie pas **tous** les gâteaux prévus. Certains ont pu être abandonnés par manque de place.

On l'a vu, la section 5, qui normalise le comportement du client, est très complexe. La section 4 l'est moins mais l'expérience a largement prouvé que les serveurs émettant des gâteaux avaient du mal à se tenir à un profil strict, et produisaient trop de variations. Une des raisons identifiées par la section 6.2 est le manque d'API. La plupart du temps, les applications font directement un `printf "Set-Cookie: foobar=1234567; Expires=Wed, 06 Jun 2012 10:18:14 GMT\n"` (ou équivalent) au lieu d'appeler une API plus abstraite dont l'implémentation se chargera de sérialiser proprement les objets complexes comme la date. Avec tant de liberté laissée au programmeur, il n'est pas étonnant qu'il y ait des variations non-standard.

Si le RFC ne propose pas de solution, il insiste sur l'importance pour les programmeurs d'utiliser des bibliothèques bien déboguées et de ne pas se croire capable d'écrire un `Set-Cookie` correct après cinq minutes de lecture de la norme. Un exemple d'une telle bibliothèque est, en Python, http.cookies <https://docs.python.org/3/library/http.cookies.html>. D'encore plus haut niveau, dans le même langage, urllib2 <http://docs.python.org/library/urllib2.html> gère les gâteaux toute seule.

Comme le respect de la vie privée est le principal problème des gâteaux, il est normal que cette question ait une section entière, la 7, qui détaille ce point. C'est inhabituel à l'IETF, où la vie privée ne génère en général que peu d'intérêt. Peu de RFC ont une section "*Privacy considerations*". Que dit-elle ? Que les gâteaux sont effectivement une technologie sensible (bien qu'il existe d'autres moyens de suivre un utilisateur mais moins pratiques donc moins dangereux) et que les plus problématiques sont les gâteaux envoyés à un tiers (c'est-à-dire à une autre organisation que celle qui gère visiblement la page, par le biais d'une discrète image insérée, par exemple). Par exemple, un service de statistiques utilisé sur de nombreux sites Web, ou bien un service de placement de publicités, peuvent permettre à leur gérant de croiser les visites faites sur plusieurs sites différents et ceci sans que l'utilisateur n'ait fait de visite explicite aux services de cette indiscrète organisation. Certains navigateurs sont, à juste titre, davantage paranoïaques lorsque le gâteau est envoyé par un tiers, par exemple en les refusant par défaut. D'autres ne traitent pas ces gâteaux différemment des autres. La question étant essentiellement politique, notre RFC 6265 ne définit pas de comportement normalisé, se contentant d'attirer l'attention sur ce point.

Peut-on demander son avis à l'utilisateur ? C'est ce qu'expose la section 7.2, consacrée au contrôle par l'utilisateur. Le RFC recommande que les clients Web fournissent des mécanismes de contrôle des gâteaux permettant de :

— Gérer les gâteaux stockés dans la magasin local, par exemple examiner les gâteaux en place, détruire tous ceux d'un certain domaine (voici comment faire dans Firefox <http://support.mozilla.com/en-US/kb/Deleting%20cookies>), etc,
— Refuser systématiquement les gâteaux,
— Proposer à l'utilisateur de refuser ou d'accepter chaque gâteau reçu, même si cela nécessite un peu plus de travail lors de la navigation (mais cela a l'avantage de faire prendre conscience de la quantité de gâteaux mis par des tiers, sur des pages Web qui semblaient pourtant innocentes ; cette option existait dans Firefox mais semble avoir disparu du panneau de contrôle des dernières versions de Firefox 3 ; on peut toujurs la régler « à la main » dans `about:config network.cookie.lifetimePolicy=1`),
— Traiter les gâteaux comme temporaires (et donc les détruire lorsqu'on quitte le programme), une option parfois nommée "*safe browsing*" (cf. « "*An Analysis of Private Browsing Modes in Modern Browsers*" <http://www.collinjackson.com/research/private-browsing.pdf> »); dans Firefox, elle peut se configurer <http://support.mozilla.com/en-US/kb/Enabling%20and%20disabling%20cookies> dans "*Privacy ; History ; Use custom setting for history*".

Voici la liste des gâteaux stockés, telle que Firefox permet de l'examiner et de la modifier : Et la question que pose le même Firefox lorsqu'on l'a configuré avec `about:config network.cookie.lifetimePolicy=1` :

Et le dialogue de configuration dans Firefox 4, montrant l'option qui permet de demander à chaque gâteau si on l'accepte :

À noter qu'il existe bien d'autres moyens de configurer la politique des gâteaux, si le navigateur ne fait pas ce qu'on veut, par exemple des extensions comme Cookie Monster <`https://addons.mozilla.org/en-US/firefox/addon/cookie-monster/`>, WebDeveloper (qui permet de voir tous les gâteaux de la page en cours, avec pour chacun « Éditer » ou « Supprimer ») ou Ghostery <`http://www.ghostery.com/`> pour Firefox. Le navigateur Dillo, quant à lui, à un mécanisme très détaillé <`http://www.dillo.org/Cookies.txt`>.

Même en dehors du cas de la vie privée, les gâteaux posent un certain nombre de problèmes de sécurité (section 8). L'utilisation de HTTPS, quoique évidemment souhaitable, ne protège pas complètement contre ces vulnérabilités.

Premier problème, l'autorité diffuse ("*ambient authority*", section 8.2). Le gâteau utilisé pour l'authentification sépare la désignation (l'URL) de l'autorisation. Le client risque donc d'envoyer le gâteau pour une ressource Web qui était contrôlée par un attaquant, une technique connue sous le nom de CSRF (il existe plusieurs moyens de convaincre un navigateur de se prêter à cette attaque, une étude bien plus détaillée des CSRF, qui a parmi ses auteurs l'auteur de ce RFC, est « "*Robust Defenses for Cross-Site Request Forgery*" <`http://www.adambarth.com/papers/2008/barth-jackson-mitchell-b.pdf`> »). Une solution à ce problème serait de traiter les URL comme une capacité, donnant accès à une ressource. L'URL serait alors le secret à transmettre pour avoir accès (le RFC ne note malheureusement pas que les URL ne restent pas secrets longtemps, par exemple certains navigateurs ou extensions des navigateurs - la Google Toolbar - transmettent les URL consultés à leur maître).

Autre problème des gâteaux, leur envoi en texte clair (section 8.3). Si on n'utilise pas HTTPS, `Set-Cookie` et `Cookie` sont transmis en clair avec leur contenu, permettant à un espion de les copier et de les utiliser (c'est cette vulnérabilité qu'exploite un programme comme FireSheep). En outre, un attaquant actif peut modifier ces en-têtes. La solution recommandée est évidemment de chiffrer toute la session, et de mettre l'attribut `Secure` aux gâteaux envoyés, pour éviter qu'ils ne soient plus tard transmis sur une session non chiffrée. Beaucoup de gros services très populaires ne sont pas accessibles en HTTPS ou bien ne le mettent pas en œuvre par défaut (souvent pour des raisons de performance, facteur crucial pour ces services qui reçoivent beaucoup de visites). Dans le cas de Gmail, c'est l'option « Général : Connexion du navigateur ; Toujours utiliser le protocole https ». Dans celui de Twitter, c'est « Toujours utiliser le HTTPS ». L'EFF mène une campagne pour systématiser cette protection <`http://www.eff.org/https-everywhere`>. À noter qu'on peut limiter (mais pas supprimer) la réutilisation du gâteau en liant celui-ci à l'adresse IP du client ; si le gâteau vient d'une autre adresse, il sera refusé. Cette méthode n'est pas si efficace qu'elle en a l'air en raison de la fréquence du partage d'adresses IP (via le NAT). Ainsi, dans un café avec WiFi, un attaquant qui utilise FireSheep n'aura pas de problèmes malgré cette protection car tous les clients sont probablement derrière la même adresse IPv4 (c'est sans doute pour cela que le RFC ne propose pas cette solution).

C'est encore pire si le gâteau contient directement de l'information utile. Par exemple, si le gâteau est `lang=en-AU`, l'espion qui écoute peut trouver la langue et le pays de l'utilisateur. En général, cette information est plutôt stockée sur le serveur et le gâteau ne contient qu'un identificateur de session, inutile si on n'a pas accès au serveur. Mais cette technique présente d'autres risques (section 8.4), par exemple celui du « choix de session » où l'attaquant se connecte au serveur, puis convainct la victime (par exemple par le biais d'un lien) d'utiliser l'identificateur de session et d'interagir avec le serveur (par exemple en y déposant des informations confidentielles). L'attaquant, qui a conservé l'identificateur de

session, pourra alors continuer la session et récupérer ces informations. (Voir un exemple <http://www.westpoint.ltd.uk/advisories/wp-04-0001.txt> et un article complet <http://www.acrossecurity.com/papers/session_fixation.pdf>.)

Autre problème de sécurité des gâteaux, la confidentialité (section 8.5) : comme les gâteaux ne sont pas spécifiques d'un port ou d'un plan (http, gopher, ftp, etc), un client risque toujours de transmettre les gâteaux d'un serveur à un autre. L'intégrité n'est pas non plus garantie (section 8.6). Si `bar.example.com` a mis un gâteau de domain de `example.com`, un autre serveur, `foo.example.com` peut parfaitement le remplacer. `Path` n'est pas une protection : il n'empêche pas le remplacement.

Enfin, les gâteaux dépendent du DNS et en héritent les vulnérabilités (section 8.7).

Une très bonne synthèse des failles de sécurité des gâteaux figure dans l'excellent article « "*HTTP cookies, or how not to design protocols*" <http://lcamtuf.blogspot.com/2010/10/http-cookies-or-how-not-to-d
html> ».

Voilà, c'est tout, il reste à enregistrer les deux en-têtes dans le registre des en-têtes <https://www.iana.org/assignments/message-headers/perm-headers.html>, ce que fait la section 9.

Contrairement aux autres RFC qui remplacent un autre RFC, ce RFC 6265 ne contient pas de description des changements par rapport aux précédents documents. Les clarifications et précisions sont sans doute trop nombreuses pour former une liste intéressante.

Merci à Jean-Baptiste Favre, bohwaz et Ollivier Robert pour leurs remarques et suggestions.