RFC 6454 : The Web Origin Concept

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 12 décembre 2011

Date de publication du RFC : Décembre 2011

https://www.bortzmeyer.org/6454.html

Chaque jour, plusieurs failles de sécurité sont annoncées frappant un site Web, ou, plus rarement, un navigateur. Ces failles proviennent souvent de l'exécution de code, par exemple Java ou JavaScript, téléchargé depuis un serveur qui n'était pas digne de confiance, et exécuté avec davantage de privilèges qu'il n'aurait fallu. Une des armes principales utilisées par la sécurité du Web, pour essayer d'endiguer la marée de ces attaques, est le concept d'**origine**. L'idée est que toute ressource téléchargée (notamment le code) a une origine, et que cette ressource peut ensuite accéder uniquement à ce qui a la même origine qu'elle. Ainsi, un code JavaScript téléchargé depuis `www.example.com` aura le droit d'accéder à l'arbre DOM de `www.example.com`, ou à ses "*cookies*", mais pas à ceux de `manager.example.net`. Ce concept d'origine semble simple mais il y a plein de subtileries qui le rendent en fait très complexe et, surtout, il n'avait jamais été défini précisément. Ce que tente de faire ce RFC.

Pour comprendre l'importance de l'**origine**, revoyons un navigateur Web en action. Imaginons qu'il n'impose aucune limite au code (Java, JavaScript, Silverlight, etc) qu'il exécute. Un méchant pourrait alors mettre du code malveillant JavaScript sur son serveur, faire de la publicité (par exemple via le spam) pour `http://www.evil.example/scarlet-johansson-pictures/`, et attendre les visites. Le navigateur récupérerait l'HTML, puis le JavaScript inclus, et celui-ci pourrait faire des choses comme lancer une DoS par connexions HTTP répétées vers `http://www.victim.example/`. C'est pour éviter ce genre d'attaques que tous les navigateurs ont la notion d'origine. Le code récupéré sur le site du méchant va avoir l'origine `www.evil.example`, et le navigateur l'empêchera de faire des requêtes HTTP vers un autre serveur que `www.evil.example`. En gros, le principe traditionnel de sécurité est « Le contenu récupéré depuis l'origine X peut interagir librement avec les ressources ayant cette même origine. Le reste est interdit. » (section 1 du RFC, pour un résumé).

Mais ce principe très simple à énoncer et à comprendre dissimule pas mal de pièges. Ce RFC va donc essayer de les mettre en évidence. À noter qu'il ne fait que fournir un cadre, ce n'est pas un protocole, et il n'y a donc pas besoin de modifier les navigateurs pour obéir à ce texte. Les détails concrets du principe d'origine vont en effet dépendre du protocole utilisé et chacun (HTML, WebSockets du RFC

6455 [1], etc) va donc devoir décliner ce principe selon ses caractéristiques propres. Par exemple, pour le futur HTML5, voici la définition <http://www.w3.org/TR/html5/origin-0.html#origin-0>.

Donc, on attaque avec la section 3, qui expose ce qu'est ce « principe de même origine » ("*Same Origin Policy*"). D'abord, la **confiance** (section 3.1) est exprimée sous forme d'un URI. Lorsque le serveur www.niceguy.example sert une page HTML https://www.niceguy.example/foo/bar.html qui contient :

```
<script src="https://example.com/library.js"/>
```

Il annonce sa confiance envers l'URI https://example.com/library.js. Le code JavaScript téléchargé via cet URI sera exécuté avec les privilèges de https://www.niceguy.example/foo/bar.html, qui a choisi de lui faire confiance.

Cette idée de « confiance par URI » peut poser des problèmes dans certains cas. Par exemple, si on utilise le STARTTLS du RFC 2817, le fait d'utiliser TLS ou pas n'apparait pas dans l'URI et la page HTML ne peut donc pas exiger TLS. Les URI de plan https: n'ont pas ce défaut, l'exigence de TLS y est explicite.

Mais il y a une question plus sérieuse : dans https://www.niceguy.example/foo/bar.html, quelle est l'**origine**, celle dont vont dépendre les autorisations ultérieures (section 3.2) ? Est-ce tout l'URI ? Ou bien seulement le FQDN www.niceguy.example ? Réponse : aucune des deux.

Utiliser tout l'URL comme origine manquerait de souplesse. Un code venu de http://www.niceguy.example/ ne pourrait pas interagir avec du contenu récupéré en http://www.niceguy.example/bar. Mais n'utiliser que le nom de domaine n'est pas génial non plus. Car cela permettrait à du code JavaScript récupéré par http://www.niceguy.example:8080/ de modifier un arbre DOM de https://www.niceguy.ex (notez que la seconde est sécurisée avec HTTPS ; ne se servir que du nom de domaine comme origine permettrait à du contenu non sûr de manipuler du contenu sûr).

Et faut-il utiliser tout le FQDN ? Cela empêche http://portal.niceguy.example/ de jouer avec http://manager.niceguy.example/ alors que les deux sites sont sous la même autorité, celle de niceguy.example. Mais le problème est que la détermination de cette autorité est difficile. Contrairement à ce que croient les débutants, elle n'est pas forcément dans les deux derniers composants du nom de domaine. Par exemple, jones.co.uk et smith.co.uk ne sont pas sous la même autorité. Et, même lorsque c'est le cas, une Université ne souhaite pas forcément que http://students.example.edu/ (consacré aux pages personnelles des étudiants) aient des droits sur http://admin.example.edu/. Bien sûr, aucun système n'est parfait. Avec le FQDN, http://example.edu/students/ mark aura des droits sur http://example.edu/grades/admin. Mais ce modèle fondé sur le FQDN est maintenant trop ancien pour le changer (rappelez-vous qu'il a évolué informellement, notre RFC 6454 ne fait que le documenter a posteriori).

Donc, pour résumer le point important de la section 3.2 : une origine est un tuple {FQDN, plan, port}. Par exemple, http://example.com/, http://example.com:80 et http://example.com/toto.html

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc6455.txt

ont tous les trois la même origine, le tuple {example.com, http, 80}. En revanche, http://example.com:81/ a une autre origine (port différent), de même que ftp://example.com (plan différent), ou http://www.example.com/ (nom différent).

Ce principe de même origine ne répond pas à tous les risques de sécurité. Il faut également intégrer la notion d'autorité (tel que ce terme est utilisé en sécurité, notez que le RFC 3986 utilise ce mot dans un autre sens). L'autorité est que le navigateur donne des droits différents selon le type de document. Une image n'a aucun droit, c'est du contenu passif (du moins pour les formats traditionnels : avec des formats comme SVG, l'analyse de sécurité est bien plus complexe). En revanche, un document HTML a le droit de déclencher le chargement d'autres ressources (styles CSS mais surtout code actif, par exemple JavaScript). Normalement, l'autorité dépend du type MIME indiqué (image/png pour une image, text/html pour l'HTML). Si ce type est déterminé par un utilisateur, de nouvelles vulnérabilités peuvent survenir (et certains navigateurs sont assez imprudents pour deviner le type MIME en examinant le contenu, une mauvaise pratique, connue sous le nom de "*content sniffing*"). Pire, si l'utilisateur peut insérer du contenu qu'il choisit dans des documents ayant une forte autorité (pages HTML), de nouveaux risques peuvent survenir. Par exemple, une attaque XSS débute par l'insertion de contenu dans une page HTML, contenu qui sera ensuite interprété par le navigateur, et exécuté avec l'origine de la page Web (cf. section 8.3). Or, ce cas est très fréquent (pensez à un moteur de recherche qui affiche la requête originale sans précaution, alorss qu'elle était choisie par un utilisateur extérieur). La protection contre les XSS est complexe mais le conseil de base est : modifiez tout contenu venu de l'extérieur avant de l'inclure dans un document ayant de l'autorité (comme une page HTML), de façon à ce qu'il ne puisse pas être interprété par le navigateur. Le plus simple pour cela est d'utiliser exclusivement un système de gabarit pour fabriquer les pages. Il se chargera alors des précautions <https://www.bortzmeyer.org/caracteres-dangereux.html> comme de transformer les ¡ en <.

Une fois déterminée l'origine, il reste à décider des privilèges donnés aux objets de la même origine (section 3.4). Pour les objets récupérés, on l'a vu, la politique est typiquement de ne donner accès qu'aux objets de même origine que le code. Mais pour les accès à d'autres objets, tout est bien plus complexe et la sécurité rentre ici en conflit avec l'utilisabilité. Par exemple, une application stricte du principe de même origine aux accès aux ressources interdirait à une page Web de mener à une autre page, d'origine différente, alors que c'est évidemment une fonction essentielle du Web. Moins caricatural, le cas d'images ou autres contenus embarqués dans une page Web. Bien que cela pose des problèmes de sécurité (pensez aux "*Web bugs*"), aucun navigateur n'impose que les images aient la même origine que la page qui les charge... De même, aucun navigateur n'interdit complètement d'envoyer des données à une autre origine, même si c'est la base de nombreuses CSRF.

Voilà, l'essentiel des principes du RFC tient dans cette section 3. Le reste, ce sont les détails pratiques. La section 4 formalise la notion d'origine en en donnant une définition rigoureuse. Elle couvre des cas rigolos comme les URI de plan file: (autrefois, les navigateurs considéraient tous les fichiers locaux comme ayant la même origine, aujourd'hui, les plus paranoïaques font de chaque fichier une origine différente), ou bien explicite le cas des ports par défaut de certains plans (vous avez remarqué l'équivalence entre les URI http://example.com/, http://example.com:80, plus haut ?). La section 5 décrit d'autres pièges comme le cas des plans où n'apparaissent pas de noms de machine comme par exemple les data: du RFC 2397 (un cas amusant car deux URI data: complètement identiques, bit par bit, sont quand même d'origine différente). Et la section 6 explique la sérialisation d'un URI, à des fins de comparaison simples avec d'autres URI, pour déterminer s'ils ont la même origine. Par exemple, http://www.example.com:80/ et http://www.example.com/toto se sérialisent tous les deux en http://www.example.com. Pour les URI de type {"*scheme*", "*host*", "*port*"} (les plus fréquents), une simple comparaison de chaînes de caractères nous dira ensuite s'ils ont la même origine (ici, oui).

Si on revient au premier exemple JavaScript que j'ai donné, celui où https://www.niceguy.example/foo/bar.ht charge https://example.com/library.js, on voit que l'origine n'est pas l'URI du script mais celle

de la page qui le charge. Or, on peut imaginer que le serveur `example.com` voudrait bien connaître cette origine, par exemple pour appliquer des règles différentes. ou tout simplement pour informer le script du contexte dans lequel il va s'exécuter. C'est le rôle de l'en-tête HTTP `Origin:`, normalisé dans la section 7, et qui peut être ajouté aux requêtes. Ainsi, dans l'exemple ci-dessus, la requête HTTP ressemblerait à :

```
[Connexion à exemple.com...]
GET /library.js HTTP/1.1
Host: example.com
Origin: https://www.niceguy.example
...
```

On voit que l'origine indiquée est la forme sérialisée comme indiqué plus haut. Ce n'est donc pas l'équivalent de `Referer:` qui indique un URI complet.

Cet en-tête est désormais dans le registre des en-têtes <https://www.iana.org/assignments/message-headers/perm-headers.html> (cf. section 9).

Comme tout ce RFC décrit un mécanisme de sécurité, il est prudent de bien lire la section 8, "*Security Considerations*", qui prend de la hauteur et rappelle les limites de ce mécanisme. Historiquement, d'autres modèles que le « principe de même origine » ont été utilisés (à noter que le RFC ne fournit pas de référence), comme le "*taint tracking*" ou l'"*exfiltration prevention*" mais pas avec le même succès.

D'abord, la notion d'origine est une notion unique qui s'applique à des problèmes très différents et peut donc ne pas être optimisée pour tous les cas.

Ensuite, elle dépend du DNS (section 8.1) puisque la plupart des plans d'URI utilisent la notion d'"*host*" et que la politique de même origine considère que deux serveurs sont les mêmes s'ils ont le même nom. Si l'attaquant contrôle le DNS, plus aucune sécurité n'est possible (on croit parler au vrai `www.example.com` et on parle en fait à une autre machine).

À noter que le RFC prend bien soin de ne **pas** parler de la principale vulnérabilité de ce concept d'"*host*" : l'absence de notion générale d'**identité** d'une machine dans l'Internet (cette notion n'existe que pour certains protocoles spécifiques comme SSH). En l'absence d'une telle notion, des attaques sont possibles sans que l'attaquant ait eu à compromettre le DNS, comme par exemple le changement d'adresse IP <https://www.bortzmeyer.org/dns-rebinding-pinning.html>.

Autre piège, le fait que le principe de même origine, ne soit pas apparu tout de suite et que certaines techniques de sécurité du Web utilisent une autre unité d'isolation que l'origine (section 8.2). Le Web n'a pas en effet de sécurité cohérente. C'est le cas des "*cookies*" (RFC 6265), qui se servent d'un autre concept, le « domaine enregistré ». Le navigateur essaie de trouver, lorsqu'il récupère un "*cookie*", à quel « domaine enregistré » il appartient (l'idée étant que `sales.example.com` et `it.example.com` appartiennent au même domaine enregistré, alors qu'ils n'ont pas la même origine). Il enverra ensuite le "*cookie*" lors des connexions au même domaine enregistré. Pour déterminer ce domaine, notez que les "*cookies*" ne font pas de différence entre HTTP et HTTPS (sauf utilisation de l'attribut `Secure`) et qu'un "*cookie*" acquis de manière sûre peut donc être envoyé par un canal non sûr.

Notre RFC note que cette pratique d'utiliser le « domaine enregistré » est mauvaise et la déconseille. En effet :

— Rien n'indique dans le nom lui-même où est le domaine enregistré. Sans connaître les règles d'enregistrement de JPRS, vous ne pouvez pas dire que `a.co.jp` et `b.co.jp` sont deux domaines enregistrés différents. Il existe des listes publiques de politiques d'enregistrement comme <http://publicsuffix.org/> mais aucune n'est officielle et aucune n'est à jour.
— Cette pratique dépend fortement du plan d'URI utilisé, certains n'ayant pas de domaine (donc pas de domaine enregistré).

Autre faiblesse du principe de même origine, l'autorité diffuse (section 8.3). L'autorité va dépendre de l'URI, pas du contenu. Or, un URI de confiance peut inclure du contenu qui ne l'est pas (pour le cas, fréquent, où une page Web inclut du contenu généré par l'utiisateur) et c'est alors un risque de XSS.

Résultat, le principe de même origine ne suffit pas à lui seul à protéger l'utilisateur (plusieurs attaques ont déjà été documentées <http://www.azizsaleh.com/index.php/ScriptsAndResources/Flex-Bypass-Same-Origin-Policy>.) Il faut donc ajouter d'autres pratiques de sécurité.

Notez que la question des mises en œuvre de ce RFC ne se pose pas, il a été réalisé après le déploiement du concept de même origine et tous les navigateurs Web utilisent aujourd'hui ce système. Toutefois, certains ne le font peut-être pas encore absolument comme décrit ici, par exemple en accordant au plan ou au port moins de poids qu'au "*host*" (il semble que cela ait été le cas de certaines versions d'Internet Explorer).

Une autre bonne lecture sur ce concept d'origine est l'article de Google pour les développeurs <http://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy>.