RFC 6943 : Issues in Identifier Comparison for Security Purposes

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 10 mai 2013

Date de publication du RFC : Mai 2013

https://www.bortzmeyer.org/6943.html

Utiliser des **identificateurs** (noms de domaine, URI, noms d'utilisateur, adresses de courrier, etc) comme clés d'accès à des informations de sécurité est courant. Par exemple, on autorise machin@truc.example et lui seul à accéder à certains contenus. Cela implique une **comparaison** entre l'identificateur présenté et ceux stockés dans une base. En apparence, rien de plus simple que de comparer deux chaînes de caractères. En réalité, il existe plein de pièges, que documente ce RFC de l'IAB. Si tout le monde n'utilise pas exactement le même algorithme de comparaison (et certains sont mal spécifiés ou mal connus et permettent donc des variations), alors on peut avoir aussi bien des dénis de service (utilisateur légitime refusé) que des augmentations de privilèges (utilisateur illégitime accepté).

L'informaticien naïf peut croire que comparer deux identificateurs, c'est simplement faire une comparaison bit à bit de leur représentation <https://www.bortzmeyer.org/representation-texte.html> mais non, c'est plus compliqué que cela.

Pour comprendre le problème, la section 1 du RFC commence par parler du cycle de vie d'un identificateur. Il est d'abord généré, par exemple par une autorité centrale. Il est ensuite souvent stocké en deux endroits, chez l'utilisateur qui va s'en servir et dans une base des identificateurs enregistrés. Par exemple, si c'est une autorité centrale qui a fabriqué l'identificateur, elle le met dans sa base de données (cas, par exemple, d'un registre de noms de domaine). Naturellement, il y a des tas de possibilités différentes. Par exemple, un identificateur peut être une clé publique cryptographique, générée localement et stockée de même.

L'identificateur est ensuite distribué à ceux et celles qui en auront besoin. Cela peut se faire par des moyens numériques mais aussi par des moyens traditionnels comme une carte de visite ou une communication téléphonique. Pensez à un URL que l'on utilise dans des publicités, dans l'espoir que des gens

tapent ensuite cet URL dans la barre d'adresse de leur navigateur. Ce passage par des moyens de communication non numériques est une des sources de problèmes car taper correctement un identificateur lu en vitesse sur le flanc d'un autobus n'est pas trivial.

Enfin, quelqu'un va utiliser cet identificateur. Par exemple, il va essayer d'envoyer un message à `barack@whitehouse.gov` et espère que cela atteindra la boîte aux lettres de quelqu'un de l'équipe du Président. Ou bien un utilisateur va récupérer un identificateur et essayer de le comparer avec celui qu'il connait. C'est le cas d'un navigateur Web voulant valider un certificat X.509 (RFC 6125[1]).

À noter qu'il existe plein d'autres complications possibles. Par exemple, une entité peut être désignée par plusieurs identificateurs (ce RFC est à la fois `http://www.ietf.org/rfc/rfc6943.txt` et `http://www.rfc` un être humain peut être référencé par son numéro de passeport ou bien par son numéro de Sécu). Notre RFC ne se préoccupe pas de ce cas, se limitant à celui, déjà assez difficile, de la comparaison de deux identificateurs pour voir s'ils sont identiques.

Cela peut se faire de trois façons (section 1.1, et voir aussi le RFC 6885 qui avait introduit ces trois cas) :

— Identificateurs **absolus** : ce sont les cas idéaux, ceux où une comparaison bit à bit convient. Les adresses IP sous forme binaire sont dans ce cas : si vous mettez `2001:db8:1::1317` dans une ACL, il n'y a aucune ambiguité pour déterminer si l'adresse présentée est égale ou non à celle-ci. (Attention, ce n'est vrai que pour la forme binaire des adresses IP, pas pour leur représentation textuelle.)
— Identificateurs **définis** : il existe un algorithme bien défini pour comparer deux identificateurs. Un cas typique est celui d'une comparaison insensible à la casse, par exemple pour les noms de domaine. (Au fait, la section 9.2.1 du RFC 4790 contient d'utiles conseils.)
— Identificateurs **indéfinis** : il n'existe pas vraiment d'algorithme pour les comparer. Les noms humains sont dans ce cas (est-ce que George Martin est la même personne que George R. R. Martin ?)

Une technique courante pour faciliter les comparaisons des identificateurs définis est la **canonicalisation**. On réduit d'abord l'identificateur à une forme canonique et on fait ensuite une comparaison absolue (bit à bit). Pour des noms de domaines, on peut par exemple toujours les passer en minuscules avant de comparer. Dans le cas d'identificateurs Unicode, c'est évidemment plus complexe mais il existe plusieurs algorithmes de canonicalisation Unicode. L'important est que toutes les parties impliquées utilisent le même.

On peut bien sûr comparer sans canonicaliser mais avoir une forme canonique est tellement pratique (par exemple pour l'affichage) que cela vaut toujours la peine d'en définir une. Ce faisant, on définit aussi un algorithme de comparaison.

La section 2 cite des exemples d'utilisation d'identificateurs dans des contextes de sécurité. Par exemple, trouver une clé en échange d'un nom ("*a principal*", dit-on en sécurité), chercher dans une ACL si une entité est autorisée, compter l'activité d'une entité donnée (et il faut donc ajouter son activité dans la bonne ligne du tableau). Le point important est qu'il faut que tout le monde utilise le même algorithme. Si on stocke l'activité d'adresses de courrier électronique sans les canonicaliser, et que quelqu'un change son adresse de `jean@durand.example` à `jean@Durand.EXAMPLE` (pourtant la même adresse), il pourra apparaître comme vierge, comme n'ayant pas d'activité précédente.

Les cas réels peuvent être très compliqués. Par exemple, en HTTPS, on compare ce qu'a tapé un utilisateur dans la barre d'adresses du navigateur avec ce que contient le certificat (RFC 6125). Plusieurs

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc6125.txt`

protocoles différents sont en jeu (de la définition des URL à celle de X.509) et plusieurs acteurs (des utilisateurs qui tapent plus ou moins bien, sur des systèmes très variés, et tout le monde des AC), chacun de ces acteurs pouvant avoir ses propres règles.

En cas d'algorithmes différents utilisés par des parties différentes, on peut avoir aussi bien des **faux positifs** que des **faux négatifs**. Les faux positifs, c'est quand deux identificateurs sont considérés comme identiques alors qu'ils ne devraient pas. (Je me souviens d'un vieux système Unix où le nom de "*login*" était silencieusement tronqué à huit caractères, donc `bortzmeyer` et `bortzmeye` étaient considérés identiques.) Si les privilèges sont attribués en fonction de cette égalité, on a un gain en privilèges. Si, par contre, les privilèges sont refusés en fonction de cette égalité (cas d'une liste noire), on a un refus d'un service légitime. Le faux négatif, c'est le contraire : deux identificateurs considérés comme différents alors qu'ils sont équivalents (cas de `jean@durand.example` et `jean@Durand.EXAMPLE` plus haut, si on oublie que le nom de domaine est insensible à la casse). Les conséquences sont opposées : si les privilèges sont attribués en fonction de cette égalité, on a un refus de service. Si, par contre, les privilèges sont refusés en fonction de cette égalité, on a un gain de privilèges, à tort.

Évidemment, le gain de privilèges est plus grave que le refus de service et c'est pour cela qu'on trouve, par exemple, dans la section 6.1 du RFC 3986 « "*comparison methods are designed to minimize false negatives while strictly avoiding false positives*" ». (Cet exemple suppose que les privilèges sont accordés en fonction de l'égalité et que les faux positifs sont bien plus graves.)

Le RFC donne un exemple où les identificateurs sont des URI. La société Foo paie `example.com` pour accéder à un service nommé Stuff. Alice, employée de Foo, a un compte identifié par `http://example.com/Stuff/`
En comparant des URI, Foo tient compte du fragment (la partie après le #, section 3.5 du RFC 3986) ce qu'`example.com` ne fait pas. Et Foo permet les # dans les noms de compte. Un autre employé de Foo, le malhonnête Chuck, se fait créer un compte avec l'identificateur `http://example.com/Stuff/FooCorp/alice#stuff`
Foo ne voit pas le problème puisque cet identificateur n'existe pas. Chuck va donc pouvoir obtenir des autorisations d'accès de Foo. Il peut ensuite se connecter auprès d'`example.com` comme étant `http://example.com/Stuff/FooCorp/alice`, l'identificateur d'Alice. Certes, l'autorisation de Chuck n'était valable que pour `http://example.com/Stuff/FooCorp/alice#stuff` mais rappelez-vous qu'`example.com` compare les URI en ignorant les fragments... Voici un cas où les différences entre les mécanismes de comparaison d'identificateurs ont permis un accroissement illégitime de privilèges.

Après cet exemple, la section 3 fait le tour des identificateurs les plus courants et de leurs pièges spécifiques. D'abord, les noms de machines. Ce sont normalement un sous-ensemble des noms de domaines (RFC 6055) mais notre RFC utilise ce terme dans un sens plus large, pour parler de tous les fois où un identificateur ou composant d'identificateur est appelé "*Host*". Ils sont souvent utilisés comme identificateurs, soit directement (par exemple dans le RFC 5280), soit indirectement, comme partie d'un identificateur (le RFC cite l'exemple des URI et des adresses de courrier). Le RFC note bien que ce terme de nom de machine ("*hostname*") est ambigu. Ainsi, dans `tyrion.lannister.got`, le nom de machine est-il `tyrion` ou bien `tyrion.lannister.got` (section 3.1 du RFC 1034) ? Cela peut entraîner des problèmes lorsqu'on veut décider si la machine `tyrion` a accès aux privilèges de la machine `tyrion.lannister.got`...

Dans le sens large qu'il a ici « nom de machine » peut aussi être une adresse IP littérale. Cela entraîne d'autres confusions possibles. Par exemple, si le TLD `.42` existe et qu'un nom `103.2.1.42` est enregistré, comment le distinguer de l'adresse IPv4 `103.2.1.42` ? Normalement, la section 2.1 du RFC 1123 règle la question : on doit tester l'adresse IP d'abord et `103.2.1.42` n'est donc jamais un nom. Mais il n'est pas sûr que tous les programmes appliquent le RFC 1123... Certaines personnes pensent donc qu'il y a un risque à accepter des TLD numériques, même si le RFC 1123 est clair.

Autre source d'ambiguité : la norme POSIX 1003.1 de l'IEEE admet pour une adresse IPv4 plusieurs formes, pas seulement la forme classique en quatre composants séparés par des points. Ainsi,

`10.0.258`, `0xA000201` et `012.0x102` sont des représentations légales de la même adresse, `10.0.1.2`. Certaines normes se tirent d'affaire en imposant la forme stricte, celle avec les quatre composants décimaux séparés par des points. C'est le cas des URI, par exemple (« *"an IPv4 address in dotted- decimal form"* »). Même chose avec `inet_pton` qui n'accepte que la forme stricte. Si les différentes formes sont acceptées, on peut avoir un problème d'ambiguité.

Et avec IPv6 ? Il y a également plusieurs représentations texte possibles (mais, à mon avis, moins susceptibles de poser des problèmes en pratique), par exemple `2001:db8::1` et `2001:DB8:0:0:0:0:0:1` pour la même adresse, sans compter des cas plus tordus comme les identificateurs de zone dans les URL (RFC 6874). Contrairement à IPv4, il existe une représentation canonique, normalisée dans le RFC 5952 mais elle n'est pas forcément utilisée par tous.

L'internationalisation (RFC 2277) ajoute évidemment quelques questions. Par exemple, la section 3.2.2 du RFC 3986 autorise un nom de domaine Unicode à être écrit en encodage pour-cent **ou** en punycode (le second étant recommandé mais pas imposé). Comment comparer `caf%C3%A9.fr` et `xn--caf-dma.fr` ? Comme souvent en matière d'internationalisation (qui n'a jamais été complètement acceptée par certains), le RFC pinaille même plus loin en imaginant le cas (purement hypothétique) d'un registre qui accepterait l'enregistrement de noms commençant par `xn--`, entrainant ainsi une confusion avec des IDN.

Autre façon de comparer des noms : les résoudre en adresses IP et comparer les adresses. C'est ce que fait la bibliothèque standard Java par défaut (classe URL <`http://docs.oracle.com/javase/7/docs/api/java/net/URL.html`>). Cette méthode a évidemment toujours été dangereuse, mais c'est encore pire maintenant, avec les adresses IP privées, les trucs du DNS pour avoir une réponse dépendant de la source, les mobiles, etc. Elle était conçue pour lutter contre des failles de sécurité comme le changement DNS <`https://www.bortzmeyer.org/dns-rebinding-pinning.html`> mais le jeu en vaut-il la chandelle ? Sans compter qu'il est contestable d'attendre le DNS juste pour comparer deux identificateurs.

Après les noms de machines, les ports. L'URL `http://www.example.com:443/` est-il égal à `http://www.exam` `https` ayant été enregistré <`https://www.iana.org/assignments/port-numbers`> (RFC 6335) comme équivalent de 443 ? (Cet exemple est facile : la seconde forme est illégale dans un URL HTTP. Mais, dans d'autres cas, cela peut être ambigu.)

On a souvent vu les URI dans les deux sections précédentes, consacrées aux noms de machines et aux ports. Le principal problème de la comparaison d'URI est qu'un URI est formé de plusieurs composants, chacun suivant des règles de comparaison différentes. Second problème, il existe plusieurs mécanismes standard de comparaison d'URI (RFC 3986, section 6.2, qui décrit l'échelle des comparaisons, de la plus simple à la plus complète). Le but de cette variété est de permettre aux diverses applications des URI d'optimiser pour les performances ou pour la sécurité. L'inconvénient est que deux comparateurs d'URI peuvent donner des résultats différents sans même qu'on puisse accuser l'un d'eux d'être bogué ou non standard.

Certains composants de l'URI posent des problèmes particuliers : les plans définissent la syntaxe spécifique d'un type d'URI et il ne faut donc jamais essayer de comparer deux URI de plans différents (`http` et `ftp` par exemple, même si, dans ce cas, la syntaxe est la même). Un autre cas souvent oublié dans les URI est la partie nommée "*userinfo*" avant le @, par exemple dans `ftp://alice:bob@example.com/bar`. Doit-elle être considérée significative en comparant des URI ? Le RFC ne fournit pas de règles à ce sujet.

Le chemin après le nom de machine pose un autre problème, celui des caractères `.` et `..` qui, normalement, indiquent un chemin relatif. Mais la section 5.2.4 du RFC 3986 fournit un algorithme pour les retirer, transformant `http://example.com/a/b/c/./../../g` en `http://example.com/a/g`. Un nouveau piège pour la comparaison ?

Continuons vers la fin de l'URI. Après le ? il y a une requête. Doit-elle être prise en compte dans la comparaison ? Là encore, pas de réponse simple, c'est à l'application de décider si `http://www.example.org/foo/bar?` est identique à `http://www.example.org/foo/bar`. Un exemple où cette question se pose est celle d'un site de référencement d'URI, avec les nombreux cas où la requête ne stocke en fait qu'une variable de suivi de la circulation de l'URI (lu sur Twitter, lu sur Facebook, etc).

Reste le fragment, la dernière partie d'un URI, après le #. En général, lorsqu'on utilise un URI comme identificateur dans un contexte de sécurité, on ignore le fragment (voir l'exemple plus haut avec Chuck et Alice...) Mais ce n'est pas une règle qui marche dans tous les cas. Là encore, l'important est la cohérence : que toutes les applications qui gèrent cet URI fassent pareil.

Comme pour les noms de machine, dans l'exemple Java plus haut, on pourrait se dire qu'une façon simple de comparer deux URI est de les déréférencer et de voir s'ils pointent vers des contenus identiques. Mais tous les URI ne sont pas déréférençables, on n'a pas forcément envie d'imposer une connexion Internet en état de marche juste pour comparer deux identificateurs et, de toute façon, un tel algorithme serait très fragile (que faire si on trouve le même document XML mais avec des encodages différents ?) En outre, toute démarche active comme celle-ci est dangereuse pour la vie privée (elle informe les gérants des serveurs Web de ce que l'on est en train de faire, comme le font les Web bugs).

Après les URI, place à une catégorie d'identificateurs très souvent utilisés pour identifier une entité, les adresses de courrier (RFC 5322 pour leur syntaxe, et RFC 6532 pour le cas où elles sont internationalisées). Une adresse de courrier, comme un URI, a plusieurs parties, qui suivent des règles différentes pour la comparaison. La partie à droite du @ est un nom de domaine et ce cas a été traité plus haut. La partie gauche, dite partie locale, est un identificateur indéfini : ses règles ne sont pas connues à l'extérieur et on ne peut donc pas savoir, par exemple, si `rms@gnu.org` et `RMS@gnu.org` sont le même utilisateur, ou si `stephane+ps@bortzmeyer.org` est le même que `stephane+ump@bortzmeyer.org`. Dans le cas où des adresses de courrier sont utilisées dans un certificat, on choisit souvent une comparaison bit à bit... qui peut donner plein de faux négatifs.

Après cette liste à la Prévert de problèmes, la section 4 de notre RFC tente une synthèse. Elle identifie quatre problèmes. Le premier est la **confusion**. Un identificateur est utilisé sans que son type soit clair. Par exemple, si je tape `telnet 1000`, est-ce l'adresse IPv4 `0.0.3.232` ou bien l'adresse IPv6 `::3:e8` ? Et si je tape `ping 10.1.2.42`, est-ce que `10.1.2.42` est un nom ou une adresse (le TLD `.42` peut exister) ?

Résoudre la confusion nécessite un algorithme clair. Dans le premier exemple ci-dessus, il n'y a pas de confusion. `100` ne peut pas être une adresse IPv6 légale (la présence ou l'absence d'un `:` suffit à les reconnaître). Le second exemple est normalement clair : l'adresse IP a priorité donc `10.1.2.42` ne peut pas être un nom même si le TLD `.42` existe. Si cette règle de précédence est respectée par les implémentations, il n'y aura pas de problèmes (identificateurs définis). Mon avis personnel est que ce RFC pinaille quand même très fort sur ce point, en s'interrogeant gravement sur des problèmes théoriquement intéressants mais extrêmement tordus et donc rares en pratique.

Deuxième problème, l'**internationalisation**. Un logiciel n'a aucun problème à comparer `google.com` et `goog1e.com` et à dire qu'ils sont différents. C'est plus difficile pour un humain (vous aviez repéré le L qui était en fait un 1 ?) Toutes les fois où un humain est impliqué, dans la saisie ou la reconnaissance d'un identificateur, ce genre d'erreur peut se produire. Comme le montre cet exemple, cela n'a d'ailleurs rien de spécifique aux chaînes de caractères Unicode. Mais ce problème est souvent cité comme argument contre l'internationalisation. Bref, le point important : la sécurité ne devrait pas dépendre d'une vérification visuelle faite par un humain. (Cf. l'article de Weber <`http://www.lookout.net/files/Chris_Weber_Character%20Transformations%20v1.7_IUC33.pdf`> et le RFC 6885.)

Problème suivant, la **portée**. Certains identificateurs ne sont pas uniques au niveau mondial. `localhost` est un bon exemple. C'est également le cas des adresses du RFC 1918. On peut aussi citer l'adresse de courrier `alice` qui, utilisée depuis une machine d'`example.com` arrivera à une Alice et, depuis une machine d'un autre domaine, à une autre. Dans ce dernier cas, la bonne solution est de toujours utiliser une adresse unique (par exemple `alice@example.com`) même dans un contexte local : l'expérience prouve que les identificateurs fuient souvent d'un domaine local vers un autre.

Enfin, dernier problème identifié par cette section 4, la **durée**. Certains identificateurs ne sont pas éternels et peuvent disparaître, ou désigner une autre entité. Par exemple, `bob@example.com` peut désigner un Bob aujourd'hui et, s'il quitte l'entreprise et qu'un autre est embauché, un Bob complètement différent dans quelques mois. C'est la même chose pour les adresses IP et des tas d'utilisateurs ont déjà souffert `<https://www.bortzmeyer.org/evaluation-adresses-ip.html>` de se voir attribuer une adresse IP qui avait une mauvaise réputation.

La section 5 résume les conseils les plus importants : se méfier des identifcateurs indéfinis, prévoir une comparaison absolue ou définie pour tous les identificateurs futurs, penser aux mécanismes pour valider les identificateurs (RFC 3696). Le gros risque restera toujours le cas où plus d'un protocole utilise un identificateur donné, car les différents protocoles n'auront pas forcément les mêmes règles de comparaison des identificateurs.