RFC 8265 : Preparation, Enforcement, and Comparison of Internationalized Strings Representing Usernames and Passwords

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 6 octobre 2017

Date de publication du RFC : Octobre 2017

http://www.bortzmeyer.org/8265.html

Ah, les plaisirs de l'internationalisation du logiciel... Quand l'informatique ne concernait que les États-Unis, tout était simple. Un utilisateur ne pouvait avoir un nom (un "*login*") que s'il était composé uniquement des lettres de l'alphabet latin (et même chose pour son mot de passe). Mais de nos jours, il n'est pas acceptable de se limiter au RFC 20[1]. Il faut que [Caractère Unicode non montré [2]][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré] ou [Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré] puissent écrire leur nom dans leur écriture et l'informatique doit suivre. Notre RFC décrit comment le faire (il remplace le RFC 7613, mais il y a peu de changements, rassurez-vous).

Mais pourquoi faut-il standardiser quelque chose ? Pourquoi ne pas dire simplement « les noms d'utilisateur sont en Unicode » et basta ? Parce que les logiciels qui gèrent noms d'utilisateurs et mots de passe ont besoin de les manipuler, notamment de les **comparer**. Si `[Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ]` essaie de se loguer, et que la base de données contient un utilisateur `[Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ]`, il faut bien déterminer si c'est le même utilisateur (ici, oui, à part la casse). C'est en général assez simple dans l'alphabet latin (ou dans le cyrillique utilisé pour les exemples) mais il y a d'autres cas plus vicieux qui nécessitent quelques règles supplémentaires.

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc20.txt`
2. Car trop difficile à faire afficher par LaTeX

Le cadre général de l'internationalisation des identificateurs est normalisé dans le RFC 8264. Notre nouveau RFC 8265 est l'adaptation de ce RFC 8264 au cas spécifique des noms d'utilisateur et des mots de passe.

Ces noms et ces mots de passe (aujourd'hui, il faudrait plutôt dire phrases de passe) sont largement utilisés pour l'authentification, soit directement (SASL `PLAIN` du RFC 4616, authentification de base de HTTP du RFC 7617), ou bien comme entrée d'une fonction de condensation cryptographique (SASL `SCRAM` du RFC 5802 ou bien authentification HTTP "*digest*" du RFC 7616). L'idée est que les opérations de base sur ces noms (et sur les mots de passe) ne surprennent pas excessivement l'utilisateur, quel que soit son pays, sa langue, sa culture. Un Français ou un Belge ne sera probablement pas trop étonné que `Dupont` soit accepté comme synonyme de `dupont` mais il le serait davantage si `dupond` l'était. Évidemment, la tâche est impossible (les utilisateurs sont tous différents) mais l'idée est de ne pas faire un système parfait mais un système qui marche la plupart du temps.

C'est là qu'intervient le cadre PRECIS ("*PReparation, Enforcement, and Comparison of Internationalized Strings*") du RFC 8264. Il évite que chaque développeur d'un système d'authentification doive lire et comprendre toutes les conséquences d'Unicode, et lui permet de s'appuyer sur une analyse déjà faite. Un exemple de piège d'Unicode (et qui montre pourquoi « on va juste dire que les noms d'utilisateur peuvent être n'importe quel caractère Unicode » est sans doute une mauvaise politique) est le chiffre 1 en exposant, U+00B9 (comme ça : « [Caractère Unicode non montré] » Si vous ne voyez rien, c'est parce que votre logiciel ne sait pas afficher ce caractère. Vous pouvez toujours regarder le source HTML pour comprendre l'exemple.). Doit-on l'autoriser ? Le mettre en correspondance avec le 1 traditionnel de façon à ce que `user[Caractère Unicode non montré ]` et `user1` soient le même nom ? Imaginez un client XMPP qui vous dise « `user[Caractère Unicode non montré ]@example.com` veut être votre copain. Je l'autorise ? » et que vous acceptiez en croyant que c'était `user1@example.com`. Bien sûr, on ne peut pas réellement parler d'« attaque » dans ce cas, une telle erreur permettrait juste de faire quelques farces mais, pour le logiciel qui vérifie une identité, des confusions seraient plus gênantes. Si les « attaques » exploitant la similitude de caractères Unicode sont surtout un fantasme d'informaticien anglophone n'ayant jamais réellement accepté l'internationalisation (plutôt qu'une réalité du terrain), il est quand même plus prudent de supprimer quelques causes de cafouillage le plus tôt possible.

(Ce RFC suggère également de séparer le nom d'utilisateur, identificateur formel et n'utilisant qu'un jeu de caractères restreint, et la description (cf. RFC 8266) qui pourrait utiliser davantage de caractères. Twitter ou Mastodon fonctionnent ainsi.)

Notre RFC compte deux sections importantes, décrivant le profil PRECIS pour les noms d'utilisateur (section 3) et les mots de passe (section 4). Commençons par les noms d'utilisateur. Un nom est une chaîne de caractères Unicode composée de parties séparées par des espaces. Chaque partie doit être une instance de `IdentifierClass` et est normalisée en NFC. Pourquoi cette notion de « parties séparées par des espaces » ? Parce que la classe `IdentifierClass` du RFC 8264 ne permet pas les espaces, ce qui est gênant pour certains identificateurs (« Prénom Nom » par exemple, cf. section 3.5). D'où la grammaire de la section 3.1 :

```
username   = userpart *(1*SP userpart)
```

qui dit « un nom d'utilisateur est composé d'au moins une partie, les parties étant séparées par un nombre quelconque d'espaces ». Une des conséquences de cette grammaire étant que le nombre d'espaces n'est pas significatif : `Jean Dupont` et `Jean Dupont` sont le même identificateur.

Chaque partie étant une instance de l'`IdentifierClass` du RFC 8264, les caractères interdits par cette classe sont donc interdits pour les noms d'utilisateurs. Une application donnée peut restreindre

(mais pas étendre) ce profil. Ces noms d'utilisateurs sont-ils sensibles à la casse ? Certains protocoles ont fait un choix et d'autres le choix opposé. Eh bien, il y a **deux** sous-profils, un sensible et un insensible (ce dernier étant recommandé). Les protocoles et applications utilisant ce RFC 8265 devront annoncer clairement lequel ils utilisent. Et les bibliothèques logicielles manipulant ces utilisateurs auront probablement une option pour indiquer le sous-profil à utiliser.

Le sous-profil `UsernameCaseMapped` rajoute donc une règle de préparation des chaînes de caractères : tout passer en minuscules (avant les comparaisons, les condensations cryptographiques, etc), en utilisant l'algorithme "*toLowerCase*" d'Unicode (section 3.13 de la norme Unicode ; et, oui, changer la casse est bien plus compliqué en Unicode qu'en ASCII). Une fois la préparation faite, on peut comparer octet par octet, si l'application a bien pris soin de définir l'encodage.

L'autre sous-profil, `UsernameCasePreserved` ne change pas la casse, comme son nom l'indique. `[Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ]` et `[Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ]` y sont donc deux identificateurs différents. C'est la seule différence entre les deux sous-profils. Notre RFC recommande le profil insensible à la casse, `UsernameCaseMapped`, pour éviter des surprises comme celles décrites dans le RFC 6943 (cf. section 8.2 de notre RFC).

Bon, tout ça est bien nébuleux et vous préféreriez des exemples ? Le RFC nous en fournit. D'abord, des identificateurs peut-être surprenants mais légaux (légaux par rapport à PRECIS : certains protocoles peuvent mettre des restrictions supplémentaires). Attention, pour bien les voir, il vous faut un navigateur Unicode, avec toutes les polices nécessaires :

- `juliet@example.com` : le @ est autorisé donc un JID (identificateur XMPP) est légal.
- `fussball` : un nom d'utilisateur traditionnel, purement ASCII, qui passera partout, même sur les systèmes les plus antédiluviens.
- `fu[Caractère Unicode non montré ]ball` : presque le même mais avec un peu d'Unicode. Bien qu'en allemand, on traite en général ces deux identificateurs comme identiques, pour PRECIS, ils sont différents. Si on veut éviter de la confusion aux germanophones, on peut interdire la création d'un des deux identificateurs si l'autre existe déjà : PRECIS ne donne que des règles miminales, on a toujours droit à sa propre politique derrière.
- `[Caractère Unicode non montré ]` : entièrement en Unicode, une lettre.
- `[Caractère Unicode non montré ]` : une lettre majuscule.
- `[Caractère Unicode non montré ]` : la même en minuscule. Cet identificateur et le précédent seront identiques si on utilise le profil `UsernameCaseMapped` et différents si on utilise le profil `UsernameCasePreserved`.
- `[Caractère Unicode non montré ]` : la même, lorsqu'elle est en fin de mot. Le cas de ce sigma final est compliqué, PRECIS ne tente pas de le résoudre. Comme pour le eszett plus haut, vous pouvez toujours ajouter des règles locales.

Par contre, ces noms d'utilisateurs ne sont **pas** valides :

- Une chaîne de caractères vide.
- `Henri[Caractère Unicode non montré ]` : le chiffre romain 4 à la fin est illégal (car il a ce qu'Unicode appelle « un équivalent en compatibilité », la chaîne « IV »).
- `[Caractère Unicode non montré ]` : seules les lettres sont acceptées, pas les symboles (même règle que pour les noms de domaine `<http://www.bortzmeyer.org/idn-symbol.html>`).

Continuons avec les mots de passe (section 4). Comme les noms, le mot de passe est une chaîne de caractères Unicode. Il doit être une instance de `FreeformClass`. Cette classe autorise bien plus de caractères que pour les noms d'utilisateur, ce qui est logique : un mot de passe doit avoir beaucoup d'entropie. Peu de caractères sont donc interdits (parmi eux, les caractères de contrôle, voir l'exemple plus bas). Les mots de passe sont sensibles à la casse.

Des exemples ? Légaux, d'abord :

— `correct horse battery staple` : vous avez reconnu un fameux dessin de XKCD <`https://xkcd.com/936/`>.
— `Correct Horse Battery Staple` : les mots de passe sont sensibles à la casse, donc c'est un mot de passe différent du précédent.
— `[Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ]` : un mot de passe en Unicode ne pose pas de problème.
— `Jack of [Caractère Unicode non montré ]s` : et les symboles sont acceptés, contrairement à ce qui se passe pour les noms d'utilisateur.
— `foo[Caractère Unicode non montré ]bar` : le truc qui ressemble à un trait est l'espace de l'Ogham, qui doit normalement être normalisé en un espace ordinaire, donc ce mot de passe est équivalent à `foo bar`.

Par contre, ce mot de passe n'est **pas** valide :

— `my cat is a by` : les caractères de contrôle, ici une tabulation, ne sont pas autorisés.

Rappelez-vous que ces profils PRECIS ne spécifient que des règles minimales. Un protocole utilisant ce RFC peut ajouter d'autres restrictions (section 5). Par exemple, il peut imposer une longueur minimale à un mot de passe (la grammaire de la section 4.1 n'autorise pas les mots de passe vides mais elle autorise ceux d'un seul caractère, ce qui serait évidemment insuffisant pour la sécurité), une longueur maximale à un nom d'utilisateur, interdire certains caractères qui sont spéciaux pour ce protocole, etc.

Certains profils de PRECIS suggèrent d'être laxiste en acceptant certains caractères ou certaines variantes dans la façon d'écrire un mot (accepter "*strasse*" pour "*stra[Caractère Unicode non montré]e*" ? C'est ce qu'on nomme le principe de robustesse <`http://www.bortzmeyer.org/principe-robustesse.html`>.) Mais notre RFC dit que c'est une mauvaise idée pour des mots utilisés dans la sécurité, comme ici (voir RFC 6943).

Les profils PRECIS ont été ajoutés au registre IANA <`https://www.iana.org/assignments/precis-parameters/precis-parameters.xml#profiles`> (section 7 de notre RFC).

Un petit mot sur la sécurité et on a fini. La section 8 de notre RFC revient sur quelques points difficiles. Notamment, est-ce une bonne idée d'utiliser Unicode pour les mots de passe ? Ça se discute. D'un côté, cela augmente les possibilités (en simplifiant les hypothèses, avec un mot de passe de 8 caractères, on passe de $10\hat{\}15$ à $10\hat{\}39$ possibilités en permettant Unicode et plus seulement ASCII), donc l'entropie. D'un autre, cela rend plus compliquée la saisie du mot de passe, surtout sur un clavier avec lequel l'utilisateur n'est pas familier, surtout en tenant compte du fait que lors de la saisie d'un mot de passe, ce qu'on tape ne s'affiche pas. Le monde est imparfait et il faut choisir le moindre mal...

L'annexe A du RFC décrit les changements (peu nombreux) depuis l'ancien RFC 7613. Le principal est le passage de la conversion de casse de la fonction Unicode `CaseFold()` à `toLowerCase()`. Et UTF-8 n'est plus obligatoire, c'est désormais à l'application de décider (en pratique, cela ne changera rien, UTF-8 est déjà l'encodage recommandé dans l'écrasante majorité des applications). Sinon, il y a une petite correction dans l'ordre des opérations du profil `UsernameCaseMapped`, et quelques nettoyages et mises à jour.