RFC 6317 : Basic Socket Interface Extensions for Host Identity Protocol (HIP)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 23 juillet 2011

Date de publication du RFC : Juillet 2011

https://www.bortzmeyer.org/6317.html

Il ne sert à rien de disposer d'un nouveau protocole si les applications ne s'en servent pas. Et, pour qu'il y ait une chance qu'elles s'en servent, il faut qu'elles aient une API standard pour cela. Ce RFC spécifie une possible API pour le protocole HIP (RFC 9063[1]), un protocole qui sépare l'adresse IP de l'**identité** d'une machine. Celle-ci est désormais une clé cryptographique et l'API permet d'établir des connexions en connaissant la clé.

Cette API ne concerne pour l'instant que le langage C. Son utilité principale commence lorsque l'identificateur HIP a été obtenu (par exemple par le DNS, cf. RFC 8005) et lorsque l'adresse IP est connue. Elle prévoit toutefois d'autres cas. Spécifique à HIP, elles pourrait toutefois être étendue dans le futur à d'autres protocoles réalisant une séparation du localisateur et de l'identificateur <https://www.bortzmeyer.org/separation-identificateur-localisateur.html>, comme SHIM6 (RFC 5533). Notez que cette API utilise déjà certaines extensions créées pour SHIM6 et spécifiées dans le RFC 6316.

À quoi ressemblent les identificateurs que manipule cette API ? Nommés HIT pour "*Host Identity Tag*", ce sont des résumés cryptographiques des clés publiques des machines (clés nommées HI pour "*Host Identity*"). Stockés sur 128 bits, ils sont, pour faciliter leur traitement, représentés sous forme d'adresses IPv6, en utilisant le préfixe ORCHID décrit dans le RFC 4843. Par exemple, 2001:10::1 est un HIT.

Certaines applications, ignorantes de ce qui se passe dans les couches basses, n'auront pas besoin d'être modifiées pour HIP (cf. RFC 5338). Les autres doivent suivre notre RFC, dont la section 1 expose les principes généraux :

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc9063.txt

— Extension à l'interface socket telle que décrite dans le RFC 3493.
— Nouvelle famille d'adresses, `AF_HIP`.
— Possibilité de se connecter à un pair dont on connait l'identité mais aussi, en mode « opportuniste » à un pair inconnu (cf. section 4.1.6 du RFC 5201).

Bien, commençons par le commencement, la résolution de noms (section 3). D'abord, le cas où l'application a un **résolveur** à sa disposition (section 3.1). Celui-ci, en échange d'un nom de domaine, fournit les informations qu'il a trouvées, typiquement dans le DNS : HIT ou HI, adresses IP et peut-être des serveurs de rendez-vous (RFC 8004). L'application a donc uniquement à appeler `getaddrinfo()` et elle récupère des HIT, qu'elle utilise pour remplir les informations transmises à la prise. Le résolveur a fourni à HIP les informations auxiliaires comme la correspondance entre un HI et des adresses IP.

Et si on n'a pas de résolveur HIP (section 3.2) ? Cette API permet à une application, de fournir quand même tous les détails nécessaires pour réussir une connexion (notamment les adresses IP associées à un HI).

La section 4 définit formellement la nouvelle API. D'abord, les structures de données <`https://www.bortzmeyer.org/ip-data-structures.html`>. Le nouveau type `sockaddr` est défini ainsi :

```
typedef struct in6_addr hip_hit_t;

struct sockaddr_hip {
          uint8_t        ship_len;
          sa_family_t    ship_family;
          in_port_t      ship_port;
          uint32_t       ship_flags;
          hip_hit_t      ship_hit;
};
```

`ship_hit` est le HIT, le résumé cryptographique de l'identité de la machine avec laquelle on communique. Physiquement, c'est une adresse IPv6. Notez qu'on n'utilise pas les identités (les HI), plus complexes à manipuler dans une API en C, car de taille variable. On peut remplir ce champ avec des constantes comme `HIP_HIT_ANY` (section 4.1.1) qui signifie qu'on exige une connexion HIP, mais avec n'importe qui.

Le résolveur prend des nouvelles options (section 4.2). La nouvelle famille `AF_HIP`, passée à `getaddrinfo()`, indique que l'application veut uniquement parler HIP et que toutes les `sockaddr` doivent donc être de ce type, quant à l'option `AI_NO_HIT`, elle indique au contraire que l'application ne veut pas entendre parler de HIP.

Une fois la connexion HIP établie, l'application peut se servir de `getsockname()` et `getpeername()` pour trouver des informations sur la machine paire (section 4.3). Elles permettent par exemple d'accéder au HIT de celle-ci (ce qui est utile si l'application acceptait n'importe quel HIT).

Si la machine dispose de plusieurs identités, le choix du HIT utilisé comme source est laissé à l'initiative du système. Si l'application veut l'influencer, elle peut se servir de mécanismes analogues à ceux du RFC 5014 : une option de `setsockopt()`, `HIT_PREFERENCES` (section 4.4), et deux valeurs possibles :

— `HIT_PREFER_SRC_HIT_TMP` : donner la préférence aux HIT temporaires (dits aussi « anonymes », cf. sections 5.1.2 et 7 du RFC 5201),
— `HIT_PREFER_SRC_HIT_PUBLIC` : donner la préférence aux HIT publics.

Si HIP_HIT_ANY signifiait qu'on exigeait une connexion HIP mais avec n'importe qui, HIP_ENDPOINT_-ANY indiquait qu'on n'avait même pas cette préférence et que des connexions non-HIP étaient possibles. Mais, après coup, on veut parfois savoir ce qu'on a obtenu et la nouvelle fonction sockaddr_is_-srcaddr() (section 4.5) permet de savoir si le pair avec lequel on s'était connecté utilisait HIP ou pas. (Il y a aussi une fonction hip_is_hit() qui teste simplement si une valeur de 128 bits est bien un HIT.)

On l'a vu, cette API permet, dans une large mesure, à une application de se connecter en HIP en ignorant presque tout de ce protocole : il suffit d'indiquer AF_HIP au lieu de AF_INET6 et ça roule. La traduction des HIT en localisateurs est notamment effectuée automatiquement, l'application ne se souciant pas de ces derniers. Toutefois, certaines applications peuvent vouloir davantage de contrôle, notamment sur le choix des localisateurs (les adresses IP). La section 4.6 est faite pour elles. Elle s'appuie sur les options SHIM6 du RFC 6316. Par exemple, l'option SHIM_LOC_PEER_PREF permet, avec setsockopt(), de définir le localisateur préféré et, avec getsockopt(), de découvrir le localisateur utilisé.

Quelques points sur la sécurité viennent compléter ce RFC, en section 6. Ainsi, l'option HIP_-ENDPOINT_ANY dit explicitement qu'on accepte des pairs non-HIP, pour les connexions entrantes ou sortantes. On peut donc se retrouver sans la sécurité associée à HIP (notamment l'authentification du pair). Les applications soucieuses de sécurité devraient donc utiliser HIP_HIT_ANY et pas HIP_ENDPOINT_-ANY.

Aujourd'hui, cette extension de l'API traditionnelle ne se retrouve pas encore dans Linux, ni dans un des BSD. Pour cela, un programme comme echoping <http://echoping.sourceforge.net/> n'a donc pas encore de possibilité d'utiliser HIP (cf. bogue #3030400 <https://sourceforge.net/tracker/?func=detail&aid=3030400&group_id=4581&atid=104581>.)