RFC 8783 : Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel Specification

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 31 mai 2020

Date de publication du RFC : Mai 2020

https://www.bortzmeyer.org/8783.html

Le système DOTS ("*Distributed Denial-of-Service Open Threat Signaling*") est conçu pour permettre la coordination des défenseurs pendant une attaque par déni de service (cf. RFC 8612 [1]). Pour cela, DOTS a deux protocoles, le protocole de signalisation, à utiliser en cas de crise, et le protocole de données, pour les temps plus calmes. Ce dernier fait l'objet de ce RFC.

Pourquoi deux protocoles ? Parce qu'il y a deux sortes de données : celles urgentes, de petite taille, transmises dans le feu de l'attaque, au moment où le réseau marche mal, c'est le domaine du protocole « signalisation », du RFC 9132. Et les données de grande taille, moins urgentes, mais qui doivent être transmises de manière fiable. Et c'est le but de notre RFC. Il y a donc deux canaux entre le client DOTS, qui demande de l'aide, et le serveur DOTS qui fournit des mécanismes d'atténuation des attaques : le canal « signalisation » du RFC 9132, et le canal « données », de notre RFC. Ce canal de données va servir, par exemple, à transporter :

— La liste des préfixes IP qu'il faudra protéger. (Notez que le serveur ne va pas forcément les accepter, il peut faire des vérifications),
— Des règles de filtrage, que le serveur appliquera en cas de crise (par exemple « laisse passer uniquement ce qui va vers le port 53 » ou bien « jette tout ce qui est à destination du port 666 », ou encore « voici une liste des préfixes vraiment importants, les seuls qu'il faudra accepter en cas de DDoS »).

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc8612.txt

Pour lire le reste du RFC, ne ratez pas la section 2, sur le vocabulaire à utiliser. Notamment, le terme d'ACL est utilisé dans un sens plus général que celui du RFC 8519.

Bon, le protocole, maintenant (section 3). Comme il n'a pas besoin de fonctionner en permanence, même dans les cas d'attaque, contrairement au protocole de signalisation, et comme il a par contre besoin de fiabilité, il va utiliser des technologies classiques : RESTCONF (RFC 8040) au-dessus de TLS (et donc au-dessus de TCP). RESTCONF utilise HTTP donc on aura les méthodes classiques, `GET`, `DELETE`, etc. Par exemple, le client DOTS qui veut récupérer des informations sur la configuration commence par faire un `GET`. Les données sont encodées en JSON (RFC 8259). Elles sont spécifiées dans un module YANG (RFC 7950) et on passe du modèle YANG à l'encodage concret en suivant les règles du RFC 7951.

La connexion entre le client DOTS et le serveur peut être intermittente (on se connecte juste le temps de faire un `GET`, par exemple) ou bien permanente, ce qui est pratique si le client faite des requêtes régulières, ou bien si le serveur pousse des notifications (section 6.3 du RFC 8040).

Ah, et j'ai parlé de client et de serveur DOTS. Comment est-ce que le client trouve le serveur ? Cela peut être fait manuellement, ou bien via la procédure de découverte du RFC 8973.

Les détails sur ce qu'on peut récupérer en DOTS ? Ils sont dans la section 4, sous forme d'un module YANG, nommé `ietf-dots-data-channel` (désormais dans le registre IANA <`https://www.iana.org/assignments/yang-parameters/yang-parameters.xml#yang-parameters-1`>). Par exemple, dans ce module, les ACL permettent d'indiquer les adresses IP et les protocoles de transport concernés, ainsi que les actions à entreprendre (laisser passer le paquet, jeter le paquet, etc), comme avec n'importe quel pare-feu. Le serveur DOTS agit donc comme un pare-feu distant. Si le serveur l'accepte, on peut aussi mettre des ACL portant sur les protocoles de la couche 4, par exemple pour filtrer sur le port.

Bon, passons maintenant à la pratique. Nous allons utiliser le serveur public de test `dotsserver.ddos-secure.` Comme le protocole de données de DOTS repose sur RESTCONF (RFC 8040) qui repose lui-même sur HTTP, nous allons utiliser curl comme client. Ce serveur public exige un certificat client, que nous récupérons en ligne <`https://github.com/nttdots/go-dots/tree/master/certs`>. Ensuite, je crée un alias pour simplifier les commandes ultérieures :

```
% alias dotsdata='curl --cacert ./ca-cert.pem --cert ./client-cert.pem --key ./client-key.pem --header "Cont
```

Et enregistrons-nous, pour indiquer un identificateur de client DOTS :

```
% cat register.json
{
  "ietf-dots-data-channel:dots-client": [
    {
      "cuid": "s-bortzmeyer"
    }
  ]
}

% dotsdata  --request POST --data @register.json https://dotsserver.ddos-secure.net/v1/restconf/data/ietf-do
201
```

Voilà, l'utilisateur `s-bortzmeyer` est enregistré (201 est le code de retour HTTP "*Created*" ; si cet identificateur avait déjà existé, on aurait récupéré un 409 - "*Conflict*"). Quand on aura fini, on pourra le détruire :

```
% dotsdata  --request DELETE https://dotsserver.ddos-secure.net/v1/restconf/data/ietf-dots-data-channel:dots-dat
204
```

Suivant la logique REST de RESTCONF, l'identité du client figure dans l'URL.

Maintenant, créons un alias pour le préfixe `2001:db8::/32` :

```
% cat create-alias.json
{
  "ietf-dots-data-channel:aliases": {
    "alias": [
      {
        "name": "TEST-ALIAS",
        "target-prefix": [
          "2001:cafe::/32"
        ]
      }
    ]
  }
}

% dotsdata --request POST --data @create-alias.json https://dotsserver.ddos-secure.net/v1/restconf/data/ietf-dot
{"ietf-restconf:errors":{"error":{"error-type":"application","error-tag":"invalid-value","error-message":"alias:
400
```

Aïe, ça a raté (400 = "*Bad request*"). C'est parce que, comme nous le dit le message d'erreur, un vrai serveur DOTS n'accepte pas qu'un client joue avec n'importe quelle adresse IP, pour d'évidentes raisons de sécurité. Il faut prouver (par un moyen non spécifié dans le RFC) quels préfixes on contrôle, et ce seront les seuls où on pourra définir des alias et des ACL. Ici, le serveur de test n'autorise que trois préfixes, indiqués dans sa documentation et dans le message d'erreur. Reprenons :

```
% cat create-alias.json
{
  "ietf-dots-data-channel:aliases": {
    "alias": [
      {
        "name": "TEST-ALIAS",
        "target-prefix": [
          "2001:db8:6401::f00/128"
        ]
      }
    ]
  }
}

% dotsdata --request POST --data @create-alias.json https://dotsserver.ddos-secure.net/v1/restconf/data/ietf-dot

201
```

Ouf, c'est bon, vérifions que l'alias a bien été créé :

```
% dotsdata --request GET --data @create-alias.json https://dotsserver.ddos-secure.net/v1/restconf/data/ietf-
{"ietf-dots-data-channel:aliases":{"alias":[{"name":"TEST-ALIAS","pending-lifetime":10078,"target-prefix":["
200
```

C'est parfait, on va essayer de demander du filtrage, maintenant. D'abord, les capacités du serveur dans ce domaine :

```
% dotsdata --request GET --data @create-alias.json https://dotsserver.ddos-secure.net/v1/restconf/data/ietf-
...
   "ipv6": {
      "length": true,
      "protocol": true,
      "destination-prefix": true,
      "source-prefix": true,
      "fragment": true
    },
    "tcp": {
      "flags": true,
      "flags-bitmask": true,
      "source-port": true,
      "destination-port": true,
      "port-range": true
    },
```

Bien, le serveur sait filtrer sur la longueur des paquets, sur les adresses IP source et destination, et il sait traiter spécifiquement les fragments. Et il sait filtrer le TCP sur divers critères, comme les ports source et destination. On va lui demander de filtrer tout ce qui vient de `2001:db8:dead::/48` :

```
% cat install-acl.json
{
  "ietf-dots-data-channel:acls": {
    "acl": [
      {
        "name": "TEST-ACL",
        "type": "ipv6-acl-type",
        "activation-type": "activate-when-mitigating",
        "aces": {
          "ace": [
            {
              "name": "TEST-RULE",
              "matches": {
                "ipv6": {
                  "source-ipv6-network": "2001:db8:dead::/48"
                }
              },
              "actions": {
                "forwarding": "drop"
              }
            }
          ]
        }
      }
    ]
  }
}

% dotsdata --request POST --data @install-acl.json https://dotsserver.ddos-secure.net/v1/restconf/data/ietf-

201
```

Parfait, le serveur a accepté notre ACL. Le paramètre `activation-type` indiquait de ne pas filtrer tout de suite mais seulement lorsqu'une atténuation serait activée (`activate-when-mitigating`). Vous avez vu les fonctions les plus importantes du protocole de données de DOTS. En cas d'attaque, il ne vous reste plus qu'à utiliser le protocole de signalisation (RFC 9132) pour utiliser alias et ACLs.

Quelques considérations de sécurité, maintenant, en section 10 du RFC. La sécurité est évidemment cruciale pour DOTS puisque, si on l'utilise, c'est qu'on a des ennemis et qu'ils nous attaquent. Les préfixes IP qu'on veut protéger ou au contraire qu'on veut filtrer sont par exemple une information confidentielle (elle pourrait aider l'attaquant). Le RFC impose donc la cryptographie pour protéger les communications, en utilisant TLS, et avec chiffrement intègre. Même avec TLS et ce chiffrement intègre, comme la couche transport n'est pas protégée, un attaquant actif pourrait perturber la communication, par exemple en injectant des paquets TCP RST (qui coupent la connexion). Il existe un mécanisme de protection, AO, normalisé dans le RFC 5925, mais il est très peu déployé en pratique.

La cryptographie ne protège pas contre un client DOTS malveillant, ou simplement mal configuré. Le serveur doit donc vérifier que le client a le droit de protéger tel ou tel préfixe IP. Autrement, un client DOTS pourrait demander le filtrage du préfixe d'un de ses ennemis. Le RFC ne spécifie pas quel mécanisme utiliser pour cette vérification, mais cela peut être, par exemple, l'utilisation des registres des RIR, qui indiquent qui est le titulaire légitime d'un préfixe.

Le protocole de ce RFC peut utiliser des noms de domaine pour configurer les ACL. Dans ce cas, le serveur doit veiller à sécuriser la résolution DNS, à la fois contre des manipulations, grâce à DNSSEC, et contre la surveillance, en utilisant du DNS chiffré comme dans le RFC 7858.

Pour les mises en œuvre de ce protocole, voyez mon article sur le RFC 9132. Mais le protocole de notre RFC étant fondé sur HTTPS, on peut dans une certaine mesure utiliser des outils existants comme, vous l'avez vu, curl.