Supervision de serveurs Web .onion (« darquenette »)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 4 octobre 2017

https://www.bortzmeyer.org/monitor-tor-onion.html

Un certain nombre de webmestres configurent leur serveur Web pour être accessible via un nom en .onion. Le but principal est de résister aux tentatives de censure. Un tel service, si on est soucieux de qualité, doit être supervisé comme n'importe quel autre service. J'explique ici comment j'ai configuré cette supervision, avec le logiciel Icinga, et quelques leçons apprises à l'occasion.

Tout site Web peut être accédé via le réseau Tor si le visiteur veut garder son légitime anonymat, ou bien s'il veut échapper à certaines censures (comme un blocage de l'adresse IP du serveur). Il lui suffit d'installer le Tor Browser et hop. Mais cette technique ne protège que le client, elle ne protège pas le serveur. Des attaques côté serveur (saisie du nom de domaine, ou bien saisie du serveur physique) peuvent couper l'accès. D'où les noms de domaines en .onion (ce qu'on appelle parfois, bien à tort, « service caché », alors que ces services sont complètement publics et que même le Googlebot y passe). Avec une telle configuration (utilisée, par exemple, par mon blog <https://www.bortzmeyer.org/blog-tor-onion.html>), le risque de censure est bien plus faible (mais pas nul, attention, vous courrez quand même des risques). Les journalistes ont inventé le terme sensationnaliste de "*darknet*" (qui désignait à l'origine tout à fait autre chose) pour ces services, et les politiciens français l'ont traduit par « Internet clandestin <https://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000035638782&dateTexte=&categorieLien=id> », une traduction d'état d'urgence...(Et vraiment grotesque puisque tout le monde, vous, moi, la NSA, Google, peut visiter ces sites.)

De tels services sont de plus en plus nécessaires, et pas seulement si on est dissident en Iran, en Chine ou en Arabie saoudite. Les pays censés être démocratiques utilisent également la censure de l'Internet comme récemment en Espagne (contre le référendum catalan et donc en partie contre le .cat) ou en France avec la pérennisation de l'état d'urgence <http://www.lemonde.fr/police-justice/article/2017/09/25/antiterrorisme-collomb-defend-le-texte-a-l-assemblee_5191273_1653578.html>.

Donc, il est tout à fait justifié de configurer des services en .onion. Mais si on veut vérifier que ces services marchent bien ? Pour les services plus traditionnels (mettons un site Web classique), on utilise un logiciel de supervision. Mettons qu'on utilise, comme moi, Icinga (mais la technique expliquée ici devrait marcher avec n'importe quel logiciel de supervision compatible avec l'API de Nagios). Le script de test des serveurs HTTP est check_http <https://www.monitoring-plugins.org/doc/man/check_http.html>, qui peut se tester depuis la ligne de commande :

```
% /usr/lib/nagios/plugins/check_http  -H www.elysee.fr
HTTP OK: HTTP/1.1 200 OK - 684876 bytes in 0.791 second response time |time=0.791354s;;;0.000000;10.000000 s
```

Il est appelé régulièrement par Icinga pour tester le serveur, et déclencher une alarme si quelque chose va mal. Ce script utilise les bibliothèques « normales » du système, et ne connait pas le TLD « spécial » `.onion` :

```
% /usr/lib/nagios/plugins/check_http  -H sjnrk23rmcl4ie5atmz664v7o7k5nkk4jh7mm6lor2n4hxz2tos3eyid.onion
Name or service not known
HTTP CRITICAL - Unable to open TCP socket
```

Si on fait tourner le client Tor sur sa machine, il exporte par défaut une interface SOCKS (paramètres `SOCKSPort` et `SOCKSPolicy` dans le fichier de configuration de Tor). Cela permet à certains programmes (ici, curl) d'accéder aux services en `.onion` :

```
% curl -v  sjnrk23rmcl4ie5atmz664v7o7k5nkk4jh7mm6lor2n4hxz2tos3eyid.onion
...
curl: (6) Could not resolve host: sjnrk23rmcl4ie5atmz664v7o7k5nkk4jh7mm6lor2n4hxz2tos3eyid.onion
```

Et avec la bonne option (Tor écoute par défaut sur le port 9050) :

```
% curl -v --socks5-hostname localhost:9050 sjnrk23rmcl4ie5atmz664v7o7k5nkk4jh7mm6lor2n4hxz2tos3eyid.onion
...
* SOCKS5 communication to sjnrk23rmcl4ie5atmz664v7o7k5nkk4jh7mm6lor2n4hxz2tos3eyid.onion:80
* SOCKS5 request granted.
* Connected to localhost (127.0.0.1) port 9050 (#0)
> GET / HTTP/1.1
> Host: sjnrk23rmcl4ie5atmz664v7o7k5nkk4jh7mm6lor2n4hxz2tos3eyid.onion
> User-Agent: curl/7.55.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Fri, 22 Sep 2017 09:31:32 GMT
< Server: Apache
< Content-Type: text/html
<
{ [3245 bytes data]
  0  425k    0  3245    0     0   3245      0  0:02:14  0:00:01  0:02:13  2414<?xml version="1.0" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
<html xml:lang="fr" lang="fr" xmlns="http://www.w3.org/1999/xhtml">
<head>
...
```

(Si vous avez un message d'erreur du genre « *"* connect to 127.0.0.1 port 9050 failed : Connection refused"* », c'est que le client Tor ne tourne pas sur votre machine - il ne s'agit pas du Tor Browser, mais d'un démon - ou bien qu'il écoute sur un autre port, vérifiez la configuration.)

Petit problème, le script `check_http` ne semble pas disposer d'une option SOCKS. Il faut donc utiliser une autre méthode, le programme torsocks <`https://gitweb.torproject.org/torsocks.git/`> qui redirige divers appels système pour que l'application parle à SOCKS en croyant parler directement au serveur HTTP. Pour curl, on aurait pu faire :

```
% torsocks curl -v sjnrk23rmcl4ie5atmz664v7o7k5nkk4jh7mm6lor2n4hxz2tos3eyid.onion
```

On peut donc lancer le script de test des "*Monitoring Plugins*" <https://www.monitoring-plugins.org/> :

```
% torsocks /usr/lib/nagios/plugins/check_http -H sjnrk23rmcl4ie5atmz664v7o7k5nkk4jh7mm6lor2n4hxz2tos3eyid.onion
HTTP OK: HTTP/1.1 200 OK - 436330 bytes in 3.649 second response time |time=3.648528s;;;0.000000;10.000000 size=
```

Maintenant que le test marche en ligne de commande, il n'y a plus qu'à configurer Icinga pour l'utiliser. On écrit un petit script shell :

```
% cat /usr/local/lib/nagios/plugins/check_http_onion
#!/bin/sh

/usr/bin/torsocks /usr/lib/nagios/plugins/check_http $*
```

Et on dit à Icinga de l'utiliser. Dans `commands.conf` :

```
object CheckCommand "http-onion" {

command = [ PluginContribDir + "/check_http_onion" ]

arguments = {
"-H" = {
value = "$http_vhost$"
description = "Host name argument for servers using host headers (virtual host)"
}
[On reprend verbatim les options de check_http, qui sont, sur une
Debian, dans /usr/share/icinga2/include/command-plugins.conf]

vars.http_address = "$check_address$"
vars.http_ssl = false
vars.http_sni = false
vars.http_linespan = false
vars.http_invertregex = false
vars.check_ipv4 = "$http_ipv4$"
vars.check_ipv6 = "$http_ipv6$"
vars.http_link = false
vars.http_verbose = false
}
```

Et dans `services.conf` :

```
apply Service for (http_vhost => config in host.vars.http_onion_vhosts) {
  import "generic-service"

  check_command = "http-onion"

  vars += config
}
```

Avec ces définitions, je peux décider de superviser un service en mettant une entrée dans le `vars.http_onion_hosts` de la machine :

```
object Host "serveur-secret" {
  import "generic-host"

  ... [Autres tests]

  vars.http_onion_vhosts["onion"] = {
    http_uri = "/"
    http_vhost = "sjnrk23rmcl4ie5atmz664v7o7k5nkk4jh7mm6lor2n4hxz2tos3eyid.onion"
    http_ssl = false
    http_timeout = 15
    }
}
```

Ou bien en créant un *"host"* spécial :

```
object Host "internet-libre" {
  import "generic-host"

  check_command = "always_true"  /* No ping in Tor */

vars.http_onion_vhosts["aeris"] = {
    http_uri = "/"
    http_vhost = "aerisryzdvrx7teq.onion"
    http_ssl = false
    http_timeout = 15
    }

vars.http_onion_vhosts["amaelle"] = {
    http_uri = "/"
    http_vhost = "ozawuyxtechnopol.onion"
    http_ssl = false
    http_timeout = 15
    }

}
```

Ainsi configuré, Icinga va régulièrement tester ces services via Tor, et prévenir de la manière habituelle :

Alors, à l'usage, est-ce que ça marche ? Oui, au sens où je suis bien prévenu lorsqu'il y a un problème et Icinga teste bien et enregistre l'historique des pannes. Moins bien au sens où cette supervision révèle que Tor n'est pas parfaitement fiable, en tout cas vu de mon ADSL : les pannes temporaires sont relativement fréquentes (ce qui est logique, la requête étant acheminée par plusieurs nœuds Tor parfois lointains, et pas toujours stables). Cela peut donc valoir la peine de configurer le logiciel de supervision pour être moins prompt à lever une alarme pour ces services. Disons en gros une ou deux pannes d'environ dix minutes par jour, ce qui ne serait pas acceptable pour un serveur Web « normal ».

Ce qui est amusant est que, parfois, pendant une panne, le Tor Browser depuis le même réseau, arrive à se connecter au service. C'est probablement parce que le Tor Browser et le démon Tor utilisé par Icinga ont établi des circuits Tor différents (passant par des nœuds différents), et qu'un seul des circuits déconne.

Mille mercis à Lunar <https://people.torproject.org/~lunar/outreach-material/presenters/lunar.html> pour m'avoir tout expliqué patiemment (et, en plus, ça marche). Et merci aussi à Amaelle et Aeris pour m'avoir permis d'utiliser leurs serveurs pour les tests. Voici les trois serveurs en `.onion` qui ont été utilisé (les liens ne marcheront que si vous regardez cet article via Tor) :

— Techn0polis, journalisme au futur extérieur <http://ozawuyxtechnopol.onion/>,
— Le site d'Aeris <http://aerisryzdvrx7teq.onion/>,
— Ce blog <http://sjnrk23rmcl4ie5atmz664v7o7k5nkk4jh7mm6lor2n4hxz2tos3eyid.onion/>.