Gestion des traps SNMP

Contexte

Simple Network Management Protocol (abrégé **SNMP**), est un protocole de communication qui permet aux administrateurs réseau de gérer les équipements du réseau, de superviser et de diagnostiquer des problèmes réseaux et matériels à distance.

Une requête SNMP est un datagramme UDP envoyée par le manager à destination du port 161 de l'agent. L'agent répond alors au manager avec la valeur demandée.

Les traps SNMP, elles, sont émises depuis les agents SNMP vers une destination (un serveur de supervision par exemple), qui entendra ces requêtes. Pour les comprendre, ce serveur devra disposer de bases de données avec l'ensemble des informations (OID et Descripteurs) des constructeurs, ces bases sont appelées MIB. Les valeurs pourront alors être interprétées par le serveur de supervision. Ce procédé est souvent utilisé dans les routeurs pour par exemple, avertir qu'un lien vient de tomber sur l'une de ses interfaces. L'intérêt des traps SNMP est donc d'envoye des « alertes » dès qu'une panne apparaît sans attendre que le serveur de supervision le détecte de lui même pendant une vérification dans le cadre d'un monitoring actif.

Il se peut donc que vous souhaitiez paramétrer Shinken pour récupérer et interpréter ces traps, nous vous proposons deux moyens, via le module WS Arbiter de Shinken, ou via le module Named Pipe (aussi utilisé de manière historique dans Nagios avec le fichier nagios.cmd, que nous allons utiliser via un fichier shinken.cmd).

Attention : les traps SNMP doivent être utilisées dans un réseau sécurisé, car comme nous l'avons dit, la communication s'effectue en UDP, ce qui peut être utilisé par des personnes malintentionnées pour faire du DDOS ou encore propager des fausses traps.

Module Named Pipe

Le module Named Pipe, via le fichier "passe plat" FIFO (First In First Out) **shinken.cmd**, va permettre à Shinken de récupérer les entrées ou commandes externes.

Mise en place du module

- S'il n'existe pas, copier le répertoire named-pipe (dans le zip ici) dans le répertoire /var/lib/shinken/modules
- Il faut ensuite rajouter la définition du module :
  - Dans le répertoire /etc/shinken/modules/
  - Copier le fichier named-pipe.cfg et modifier les droits :

```
chown shinken:shinken named-pipe.cfg
chmod 644 named-pipe.cfg
```

- Ensuite on va accrocher le module au receiver
  - Ouvrir le fichier /etc/shinken/receivers/receiver-master.cfg
  - Ajouter named-pipe à la ligne modules
    - *modules named-pipe*
- Redémarrer Shinken via la commande *service shinken restart*

Création de l'élément de supervision en configuration

- Dans l'interface de configuration, il faut à présent créer le check à associer à un hôte, en passif, non actif, et volatile, car si nous souhaitons recevoir une notification dès un changement d'état. L'expiration de l'état de fraîcheur doit également être paramétrée.
  - Voici un exemple avec la syntaxe via fichiers cfg, modèle de check dédié au modèle d'hôte "TRAP-modele" :

```
define service{
        service_description     TRAP
        check_command           check-host-alive
        host_name               TRAP-modele
                is_volatile             1
        passive_checks_enabled  1
                active_checks_enabled           0
                check_freshness                         1
                freshness_threshold             300
        register                0
        check_interval              1
        retry_interval              1
}

define host{
        name                                    TRAP-modele
        register                        0
}
```

- Appliquer le modèle d'hôte à un hôte accessible sur le réseau (le paramètre "adresse" doit être rempli), par exemple un hôte "test-trap"

Rappels sur le fonctionnement des checks passifs

La configuration de check présentée ci-dessus est celle d'un check passif. Le statut d'un check de ce type va être mis à jour manuellement via la réception de traps SNMP.

Si aucun statut n'est reçu de manière passive pour ce check au bout de 5mn, Shinken déclenche une vérification manuelle pour éviter d'avoir un statut trop ancien et non représentatif de l'état de l'élément représenté par le check. Ce comportement est configuré via les options "check_freshness" (pour l'activation de ce comportement), "check_command" (pour la commande de vérification lancée par Shinken) et "freshness_threshold" (en secondes, 300 pour 5mn).

Par contre, ce comportement peut entraîner des changements de statuts du check si l'intervalle d'envoi des statuts vers le check (traps SNMP dans notre exemple) est supérieur à l'intervalle définit par l'option "freshness_threshold". Il faut donc avoir conscience de ce comportement et régler la valeur de l'option "freshness_threshold" ou bien désactiver cette fonctionnalité pour ce check ("check_freshness 0").

Script interpréteur des traps

- Maintenant il faut un script (plugin) qui va se charger d'interpréter les futures traps SNMP reçues pour les envoyer à Shinken (à travers le module named-pipe et le fichier shinken.cmd).
- Ajouter le script suivant que l'on appellera submit_check_result dans le dossier des plugins Shinken (/var/lib/shinken-user/libexec/):

```
#!/bin/bash

# Arguments:
# $1 = host_name (Short name of host that the service is associated with)
# $2 = svc_description (Description of the service)
# $3 = return_code (An integer that determines the state of the service check, 0=OK, 1=WARNING, 2=CRITICAL,
3=UNKNOWN).
# $4 = plugin_output (A text string that should be used as the plugin output for the service check)

echocmd="/bin/echo"

CommandFile="/var/lib/shinken/shinken.cmd"

# get the current date/time in seconds since UNIX epoch
datetime=`date +%s`

# create the command line to add to the command file
cmdline="[$datetime] PROCESS_SERVICE_CHECK_RESULT;$1;$2;$3;$4"

# append the command to the end of the command file
`$echocmd $cmdline >> $CommandFile`
```

- On le rend exécutable et on le donne à l'utilisateur Shinken :

```
chown shinken:shinken /var/lib/shinken-user/libexec/submit_check_result
chmod +x /var/lib/shinken-user/libexec/submit_check_result
```

- Pour tester le script et simuler une réception d'un trap translaté au format Shinken, il suffit d'exécuter la commande suivante qui va faire passer le service en état critique :

```
/var/lib/shinken-user/libexec/submit_check_result test-trap TRAP 2 "test envoi trap - CRITIQUE"
```

Les arguments sont:

- $1 = Le nom de la machine concerné par la trap
- $2 = Le nom du service (doit correspondre au nom donnée dans la définition du service Shinken. dans mon exemple: TRAP)
- $3 = Le code de retour (0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN)
- $4 = Un message texte correspondant à la sortie de la commande.

- Le check devrait passer en critique, et si au bout de la période du seuil de fraîcheur, aucun nouveau trap n'a été reçu, alors la commande check-host-alive fera repasser le check à OK (si bien sur l'hôte est accessible via le réseau).

Installation des paquets SNMPD, SNMPTRAPD et SNMPTT

Bien entendu, il faut un serveur SNMP capable de capturer les traps, ainsi que le paquet qui va les translater vers Shinken. Voici les étapes à suivre

- Installation des paquets et dépendances :

```
yum install net-snmp net-snmpd-utils net-snmp-perl snmptt perl-Sys-Syslog
```

- Note : le paquet net-snmp est pour la partie serveur, et net-snmpd-utils est pour la partie cliente afin de diagnostiquer plus rapidement des problèmes SNMP sur vos serveurs. Mais ce n'est pas obligatoire. Les paquets net-snmp-perl et snmptt permettront de translater les traps.
- Pour activer le démarrage des services SNMP au démarrage du serveur avec la commande suivante :

```
chkconfig --level 3 snmpd on;chkconfig --level 3 snmptrapd on
```

- Démarrer les services snmpd et snmptrapd :

```
service snmpd start;service snmptrapd start;service snmptt start
```

- Vous pouvez alors tester de passer une requête via les outils clients SNMP, sur son propre serveur pour vérifier la bonne communication :

```
snmpwalk -v 1 -c public -O e 127.0.0.1
```

La commande doit retourner une réponse comme par exemple :

```
SNMPv2-MIB::sysDescr.0 = STRING: Linux Shinken 2.6.32-504.16.2.el6.x86_64 #1 SMP Wed Apr 22 06:48:29 UTC 2015
x86_64
SNMPv2-MIB::sysObjectID.0 = OID: NET-SNMP-MIB::netSnmpAgentOIDs.10
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (31359) 0:05:13.59
SNMPv2-MIB::sysContact.0 = STRING: Root <root@localhost> (configure /etc/snmp/snmp.local.conf)
SNMPv2-MIB::sysName.0 = STRING: shinken
SNMPv2-MIB::sysLocation.0 = STRING: Unknown (edit /etc/snmp/snmpd.conf)
SNMPv2-MIB::sysORLastChange.0 = Timeticks: (64) 0:00:00.64
SNMPv2-MIB::sysORID.1 = OID: SNMP-MPD-MIB::snmpMPDMIBObjects.3.1.1
SNMPv2-MIB::sysORID.2 = OID: SNMP-USER-BASED-SM-MIB::usmMIBCompliance
SNMPv2-MIB::sysORID.3 = OID: SNMP-FRAMEWORK-MIB::snmpFrameworkMIBCompliance
SNMPv2-MIB::sysORID.4 = OID: SNMPv2-MIB::snmpMIB
SNMPv2-MIB::sysORID.5 = OID: TCP-MIB::tcpMIB
SNMPv2-MIB::sysORID.6 = OID: IP-MIB::ip
SNMPv2-MIB::sysORID.7 = OID: UDP-MIB::udpMIB
SNMPv2-MIB::sysORID.8 = OID: SNMP-VIEW-BASED-ACM-MIB::vacmBasicGroup
(...)
```

Interpréteur de trap SNMP

Maintenant que l'on est capable de recevoir des trap, il faut les interpréter pour dialoguer ensuite avec Shinken. Ceci s'effectue avec le paquet snmptt :

- On active le debug dans le fichier **/etc/snmp/snmptt.ini**, histoire de voir les traps que l'on a reçues mais pas interprétées. Si vous avez ma configuré quelque chose elle arriveront dans le fichier **/var/log/snmptt/snmpttunknown.log**.

```
unknown_trap_log_enable = 1
```

- On ajoute ensuite les ligne suivantes dans la configuration sous **/etc/snmp/snmptrapd.conf:**

```
disableAuthorization yes
donotlogtraps  no
traphandle default /usr/sbin/snmptt
```

Ce qui veut dire :

- on accepte toutes les interruptions
- on ne journalise pas les interruptions reçues (c'est SNMPTT qui s'en chargera)
- on traite de la même façon toutes les interruptions : avec SNMPTT

- Le service snmptrapd doit être modifié pour ne pas traduire les OID, mais les laisser sous forme numérique. C'est SNMPTT qui se chargera de cette traduction. Sous CentOS, il faut éditer le fichier **/etc/sysconfig/snmptrapd** et ajouter l'option comme ceci:

```
OPTIONS="-Lsd -p /var/run/snmptrapd.pid"
```

- Il faut activer également le processus de capture de trap nommé snmptrapd. Ceci se fait dans le meme fichier :

```
export MIBS=/usr/share/mibs
TRAPDRUN=yes
```

- On relance les deux services pour prendre en compte les modifications :

```
service snmpd restart
service snmptt restart
```

Compilation de MIB

SNMPTT a besoin de compiler les MIBs du format TXT vers un fichier de configuration. Les MIBs de CentOS se trouvent dans le dossier **/usr/share/snmp/mibs/**. Si vous voulez travailler avec une MIB particulière (routeur CISCO ou autre équipement) il vous faudra l'importer sur le système afin de la compiler. Cette compilation effectue l'association OID/ action à effectuer.

Pour chaque MIB, il faut compiler à l'aide de la syntaxe suivante :

```
snmpttconvertmib --in=<fichier MIB> \
--out=/etc/snmp/snmptt.conf.<equipement> \
--exec='/var/lib/shinken-user/libexec/submit_check_result $r TRAP 2'
```

Test avec une MIB perso

On va créer une MIB de test pour pouvoir tester toute la chaîne depuis la capture de la trap jusqu'au traitement de celle ci par Shinken:

- On créé notre mib dans un fichier sous **/usr/share/snmp/mibs/TRAP-TEST-MIB** avec le contenu suivant:

```
TRAP-TEST-MIB DEFINITIONS ::= BEGIN
IMPORTS ucdExperimental FROM UCD-SNMP-MIB;
demotraps OBJECT IDENTIFIER ::= { ucdExperimental 990 }
demo-trap TRAP-TYPE
STATUS current
ENTERPRISE demotraps
VARIABLES { sysLocation }
DESCRIPTION "Trap received !"
::= 17
END
```

- On export cette MIB

```
export MIBS=+/usr/share/snmp/mibs/TRAP-TEST-MIB
```

- **On la compile en précisant notre plugin submit_check_result, c'est donc ici que ce fait la jonction avec Shinken:**

```
snmpttconvertmib --in=/usr/share/snmp/mibs/TRAP-TEST-MIB \
--out=/etc/snmp/snmptt.conf.test \
--exec='/var/lib/shinken-user/libexec/submit_check_result $r TRAP 2'
```

- Ce qui doit générer un fichier de cette forme :

```
EVENT demo-trap .1.3.6.1.4.1.2021.13.990.0.17 "Status Events" Normal
FORMAT Trap received ! $*
EXEC /var/lib/shinken-user/libexec/submit_check_result $r TRAP 2 "Trap received ! $*"
SDESC
Trap received !
Variables:
  1: sysLocation
EDESC
```

- On fait prendre en compte ce fichier à la configuration globale de SNMPTT à la fin du fichier **/etc/snmp/snmptt.ini** en rajoutant notre fichier généré snmptt.conf.test :

```
snmptt_conf_files = <<END
/etc/snmp/snmptt.conf
/etc/snmp/snmptt.conf.test
END
```

- On relance les services :

```
service snmptt restart
service snmpd restart
service snmptrapd restart
```

- Et enfin, on lance une trap de test à la main :

```
snmptrap -v 1 -c public 127.0.0.1 TRAP-TEST-MIB::demotraps localhost 6 17 '' SNMPv2-MIB::sysLocation.0 s
"It's a trap"
```

- Les logs de snmptt doivent remonter la trap:

```
tail /var/log/snmptt/snmptt.log
Sun Feb  1 16:02:13 2015 .1.3.6.1.4.1.2021.13.990.0.17 Normal "Status Events" localhost - Trap received It's
a trap
```

- Même chose pour les log messages du système:

```
tail /var/log/messages
Feb  1 16:02:13 shinken snmptt[0]: .1.3.6.1.4.1.2021.13.990.0.17 Normal "Status Events" localhost - Trap
received It's a trap
```

- Le service passe alors en critique sur Shinken et vous recevez une notification.

Au bout de une minute, comme défini dans le fichier de configuration sur Shinken, le service passe de nouveau au vert en statut OK.

Vous avez plus qu'à importer vos MIBs et faire la liaison avec Shinken de la même façon que dans cet exemple.

Pour résumer, nous avons bien la chaîne suivante:

SNMPD (pour l'écoute des Traps) –> SNMPTT (pour la translation des Traps) –> Script (pour le passage au format Shinken) –> FIFO shinken.cmd (pour l'envoi de l'état du check de l'hôte dans Shinken)