2025/2026

Amélioration du système de supervision LibreNMS grâce à l'outil Oxidized

ANS INFORMATIQUE

2 Rue Lavoisier 25000, Besançon

Responsable Professionnel : CURTO Julien

Responsable Pédagogique : VANSTRACEELE Christophe

Stage effectué du 22/04/2025 au13/06/2025

UNIVERSITÉ MARIE & LOUIS PASTEUR

DEPARTEMENT RESAUX ET TELECOMMUNICATION
4 PLACE LUCIEN THARRADIN, 25200 MONTBELIARD

Rapport de stage de Gabin BECU

BUT R&T 2ème Année

Table des matières

Présentation de l’Entreprise

Ans informatique est une PME d'une vingtaine d'employés répartis entre les services technique, commercial et administration. Fondée en 1995, elle fait partie du groupe MCH, un groupe de PME. ANS informatique est situé 2 Rue Lavoisier au même titre que les autres entreprises du groupe MCH : ISO Telecom et SIMAT.

Cette entreprise est spécialisée dans le déploiement de matériel et logiciel informatique. Elle assure des missions telles que l’hébergement de mail, le stockage de données et de sauvegardes, la gestion de serveur virtuel, le partage de documents ...

En lien avec son partenaire, ISO Télécom ANS propose des solutions de téléphonie et réseaux par le biais d'infrastructures réseaux complètes pour des entreprises avec leur supervision et leur maintenance

Durant mon stage j'ai été intégré dans le service technique qui est composé de plusieurs techniciens responsables de la maintenance et de l'installation des équipements.

J’ai participé à l'amélioration de leur système de supervision des liens réseaux via LibreNMS.

Mon Rôle a été d’implémenter l'outil Oxidized dans leur supervision.

Mise en contexte de la supervision :

Pour superviser les liens qui nous permettent l’accès au routeur on utilise un logiciel web appelé LibreNMS qui permet de garder un visuel sur l'ensemble de ces appareils en un seul endroit. Prenons par exemple un routeur, LibreNMS va l’interroger grâce au protocole SNMP, pour lui demander plusieurs informations comme le trafic l’utilisation de la RAM, du CPU etc… Ces informations vont être recensées dans LibreNMS sous forme de graphiques ou d'informations ; Il va aussi envoyer des alertes s’il détecte des problèmes.

Figure 1: page d'un périphérique sur LibreNMS

Dans le cas d'ANS il y a 4 types de routeur. Les plus communs, environ la moitié des routeurs, sont de la marque Mikrotik avec comme système d'exploitation RouterOS. Mais il existe aussi d'autres routeur avec deux autres systèmes d'exploitation PfSense et FirewareOS. Enfin, il y a des routeurs un peu spéciaux dont on n'a pas de contrôle direct car nous sommes uniquement responsables du lien et où le SNMP n'est pas activé et donc LibreNMS se contente de ping et d'alerter si le ping ne passe plus.

Mon travail a été d'ajouter une fonctionnalité supplémentaire à cette interface grâce à Oxidized. C'est un outil qui a pour objectif de complémenter LibreNMS en récupérant la configuration des routeurs et l'ajouter dans la page de l'appareil LibreNMS au même titre que la page ou les informations de santé du routeur sont affiché.

Premiers pas

Pour commencer j'ai déjà effectué une première phase de recherche pour comprendre ce que faisait les outils et comment ils le faisaient.

Après avoir vu la théorie, Je devais passer à la pratique mais pas directement sur l'interface utilisée tous les jours par l'équipe. J'ai donc créé une VM de Test ou j'ai installé LibreNMS et ou j'ai installé Oxidized je voulais tester premièrement quelles données vont être traitées par Oxidized. J'ai donc utilisé une commande « curl » :

curl -H 'X-Auth-Token:token-trouvé-dans-LibreNMS ' http://127.0.0.1/api/v0/oxidized

Ce qui m'a donné un résultat sous forme JSON :

```
[
      {
            «  hostname «  :  « x »,
            « os » : « x »
            « ip » : « x.x.x.x »
            « group » : « x »
      }
]
```

J'ai donc créé un fichier manuellement avec les caractéristiques d'un routeur en suivant ce format.

Je devais maintenant décider avec quel protocole j'allais communiquer avec mon routeur SSH, Telnet, API… Nous avons décidé d'utiliser SSH pour sa sécurité élevée. Nous avons créé un groupe et un utilisateur qui était accessible que depuis une IP en lecture et en SSH sur un routeur Mikrotik pour maximiser la sécurité. L'inconvénient de cette méthode est que tous les routeurs ont un compte avec le même nom d'utilisateur et le même mot de passe c'est pour cette raison que l'on maximise tous les autres facteurs de sécurité

Voici le fichier « config » à l'installation d'Oxidized :

```
---
username: admin
```

```
password: password
model: junos
resolve_dns: false
interval: 3600
use_syslog: false
debug: false
threads: 30
timeout: 20
retries: 3
prompt: !ruby/regexp /^([\w.@-]+[#>]\s?)$/
rest: 0.0.0.0:8888
next_adds_job: false
vars:
  enable: enable_password
groups: {}
models: {}
pid: /home/oxidized/.config/oxidized/pid
crash:
  directory: /home/oxidized/.config/oxidized/crashes
  hostnames: false
log: /home/oxidized/.config/oxidized/logfile.log
input:
  default: ssh
  debug: false
  ssh:
    secure: false
output:
  default: file
  file:
    directory: "/home/oxidized/.config/oxidized/configs"
source:
  default: csv
  csv:
    file: "/home/oxidized/.config/oxidized/router.db"
    delimiter: !ruby/regexp /:/
    map:
      name: 0
      model: 1
      username: 2
      password: 3
```

```
    vars_map:
      enable: 4
model_map:
  cisco: ios
  juniper: junos
```

Après quelques essais et recherches, j'ai conclu que je devais modifier plusieurs lignes et que certaines était inutiles. Je suis donc arrivé à une nouvelle version plus simplifiée en enlevant le superflu et en modifiant ce qui était nécessaire :

```
---
username: XXX
password: XXX
interval: 3600
use_syslog: false
log: /home/oxidized/.config/oxidized/logfile.log
debug: false
threads: 30
timeout: 20
retries: 3
prompt: !ruby/regexp /^"([\[\]A-Za-z0-9\(\)\@\/\.\-]+?)[\]>]\z/
rest: 0.0.0.0:8888
vars:
  remove_secret: false
source:
  default: jsonfile
  jsonfile:
    file: "/home/oxidized/.config/oxidized/routerdb.json"
    delimiter: !ruby/regexp /:/
    map:
      name: name
      ip: ip
      model: os
      group: group
model_map:
  routeros: routeros
input:
  default: ssh
  ssh:
    secure: false
output:
```

```
  default: file
  file:
    directory: /home/oxidized/.config/oxidized/configs
model_map :
      routeros: routeros
```

Avec cette nouvelle version, Oxidized récupère toutes les heures les données dans le JSON, se connecte en SSH avec les identifiants, récupère la configuration. Ensuite crée un fichier dans le répertoire « configs » avec comme nom le « hostname » dans le JSON.
Mais en analysant la configuration et le fichier model pour RouterOS crée par Oxidized j'ai remarqué qu'il y avait des lignes inutiles qui étaient liées à deux commandes exécutées par Oxidized « /system routerboard print » et « / system history print without-paging » ces commandes sont déterminées par un fichier utilisé pour les routeurs sous RouterOS. J'ai donc commenté les lignes où ces commandes étaient exécutées et ces informations superflues ont disparu.

```
+# Flags: U - undoable, R - redoable, F - floating-undo
+#   ACTION                                   BY              POLICY
+# U system identity changed                  isotelecom      write
+# U system identity changed                  isotelecom      write
+# U system identity changed                  isotelecom      write
+# U system identity changed                  isotelecom      write
+# U system identity changed                  isotelecom      write
+# U system identity changed                  isotelecom      write
+# U system identity changed                  isotelecom      write
+# U system identity changed                  isotelecom      write
+# U system identity changed                  isotelecom      write
```

Figure 2 : Exemple d'une partie des ligne inutiles

Le rendre accessible dans LibreNMS

Je dois maintenant rendre cela accessible dans LibreNMS. Pour cela je dois ajouter mon routeur dans l'interface. Mais un problème intervient sur le routeur que j'utilise, le SNMP est configuré pour n'être accessible que depuis l'IP du LibreNMS or étant sur une VM je n'avais pas la même IP je ne pouvais donc pas récupérer toutes les informations de santé. J'ai donc utilisé le même système que sur les routeurs ou un technicien n'a pas d'accès en utilisant uniquement un ping. Je dois aussi changer la source d'information : j'ai donc réutilisé le token comme dans la commande « curl » vu au début.

```
source :
  default : http
  http :
    url : « http ://127.0.0.1 /api/v0/oxidized »
    map :
      name : hostname
      model: os
      ip : ip
      group: group
    headers:
```

```
X-Auth-Token : token-trouvé-dans-LibreNMS
```

Et avec ces ajouts dans la configuration : première victoire, la configuration était consultable dans LibreNMS.

Passage sur l'environnement de production

Sur ce nouvel environnement Oxidized était déjà installé mais non fonctionnel. Et LibreNMS contenait beaucoup plus de routeurs. J'ai donc avant tout comparé les configurations de LibreNMS et de Oxidized puis j'ai fait un backup. Enfin j'ai essayé de mette mon fichier et après quelques ajustements, j'avais la configuration sur mon router depuis le compte admin. Mais pas depuis le compte du support qui était en lecture seule. L'onglet « config » disparaissait, problème que je n'avais pas sur ma VM où je n'avais pas créé d'autre user que l'user admin.
J'ai donc essayé de créer un rôle qui donnait accès à l'onglet config sans succès. J'ai alors cherché une autre solution et j'ai réussi en changeant un fichier PHP dans LibreNMS à donner les droits sur l'onglet.

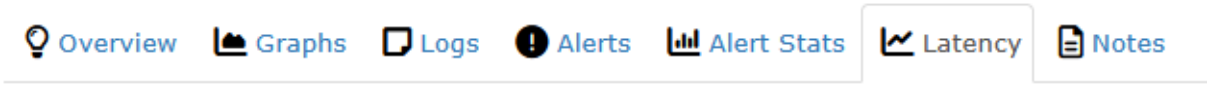

Figure 3 : onglet « config » invisible

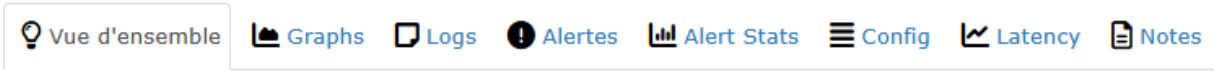

Figure 4 : onglet « config » invisible

Avoir la configuration actuelle était pratique mais une des fonctionnalité particulièrement intéressante d'Oxidized est l'historique de configuration. En utilisant git au lieu de créer des fichiers dans le dossier « configs », il est possible d'obtenir un menu déroulant avec la date de la configuration. On peut également voir la différence entre les configurations

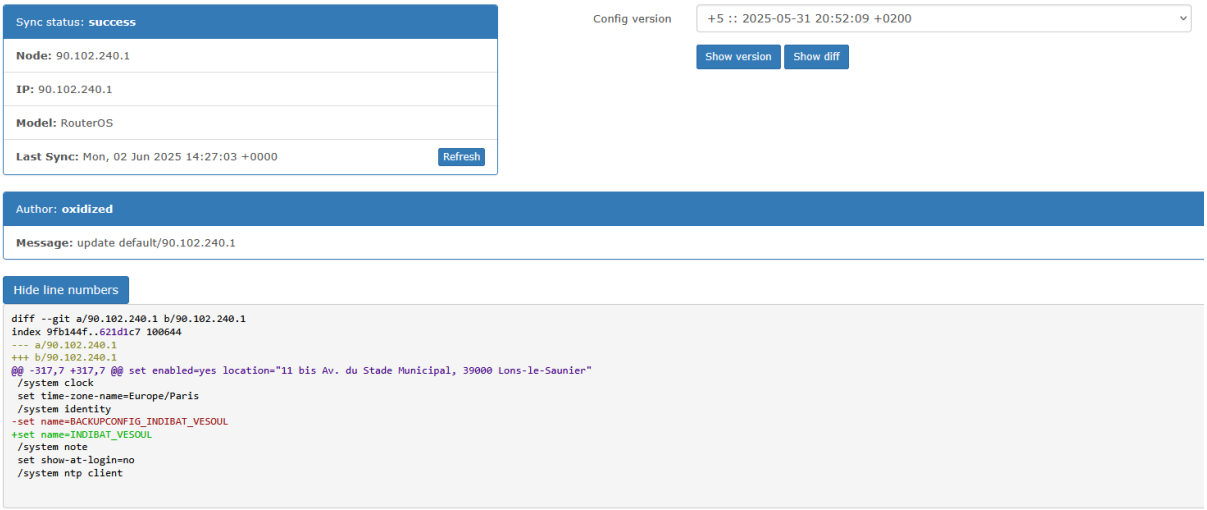

Figure 5 : Page Config avec la fonctionnalité de voir la différence entre 2 configurations

Pour cela j'ai changé les lignes de la section output :

```
output:
  default: git
  git:
    user: oxidized
    email: git@isotelecom.com
    repo: "/var/lib/oxidized/devices.git"
```

Avec ce déploiement j'avais une configuration fonctionnelle qui me permettait d'avoir un suivi de l'historique des configurations de routeur sous routerOS mais les techniciens travaillent aussi avec d'autres systèmes d'exploitation je devais donc maintenant déployer l'outil sur d'autres systèmes.

Nouvelle compatibilité de système d'exploitation

Comme mentionné plus tôt, il y a 3 systèmes d'exploitation. J'ai validé pour RouterOS, mais il serait intéressant d'avoir la même chose sur tous les appareils. J'ai commencé par FirewareOS. J'ai créé un groupe et un user sur un WatchGuard avec les droits en SSH lecture. Puis dans mon fichier de config ajouter la ligne FirewareOS dans la section « model_map » mais le problème rencontré est le port de communication SSH. Sur les appareils WatchGuard ils n'utilisent pas le port 22 mais le port 4118 pour remédier à cette différence de port. J'ai eu l'idée de créer des groupes sur Oxidized pour cela j'ai dû ajouter deux lignes dans le fichier config.php qui affectait les équipements avec l'OS « FirewareOS » dans le groupe « fireware» puis j'ai vérifié avec un curl que les groupes ont changé et j'ai ajouté cela à la fin du fichier de configuration Oxidized :

```
groups:'"
  fireware:
    vars:
      ssh_port: 4118
  default:
    vars:
      ssh_port: 22
```

Puis j'ai ajouté aussi PfSense qui lui n'avait pas de problème de port.
Ainsi, j'ai couvert l'ensemble des types de routeurs sur lequel un technicien peut avoir un accès car les routeurs en ping uniquement ne peuvent pas être paramétrés, par conséquent on ne peut pas ajouter d'utilisateur pour la connexion SSH. Cependant il reste toujours un problème, le déploiement de l'utilisateur de backup sur tous les routeurs afin qu'Oxidized puisse récupérer toutes les configurations pour cela je n'ai pas trouvé de moyen d'automatiser ce processus j'ai donc créé une procédure (confidentielle) pour ajouter un routeur et un document explicatif de comment fonctionne Oxidized (voir annexe 1) afin que même après mon départ l'équipe d'ANS Informatique puisse continuer à alimenter Oxidized

En quoi Oxidized va être très utile ?

Cet outil va désormais être une aide au quotidien dans le travail des techniciens. En effet, la mise en place d'une supervision efficace repose sur une bonne configuration des équipements. Grâce à cet outil, les routeurs Watchguard, pfSense et Mikrotik sont correctement intégrés dans l'outil de supervision, Il est essentiel de maintenir ces configurations à jour afin de pouvoir restaurer rapidement un équipement en cas de panne ou de dysfonctionnement.

Remerciements

Je tiens à remercier **Julien Curto**, Directeur Technique & Projets, pour m'avoir accueilli au sein de son service et m'avoir offert l'opportunité de réaliser ce stage dans un environnement aussi enrichissant, au sein d'une équipe bienveillante.

Je remercie également **Anthony Devaux**, alternant administrateur systèmes et réseaux, pour le temps qu'il m'a consacré ainsi que pour ses explications claires et précieuses sur de nombreuses notions essentielles à la réussite de mon projet de stage.

Annexe 1

Explication fonctionnement Oxidized

Table des matières

1. Qu'est-ce que c'est

Oxidized est un outil complémentaire à LibreNMS qui permet d'obtenir la configuration des routeurs directement dans LibreNMS.

2. Comment ça fonctionne

Oxidized fonctionne grâce à plusieurs fichiers. Parmi ceux-ci, Il y a des fichiers importants situés dans
`/var/lib/gems/version-de-gems/gems/oxidized-version/lib/oxidized/model`
Ce dossier contient les fichiers utilisés en fonction de l'OS du routeur sur lequel il essaie de se connecter et c'est ce fichier qui va exécuter les commandes sur le routeur.
Mais le fichier le plus important est celui-ci :
`/home/oxidized/.config/oxidized/config`
Ce fichier contient toute la configuration d'Oxidized.

Oxidized commence par récupérer les informations en entrée du LibreNMS grâce au token. Ces informations sont reçues sous forme d'un JSON :

```
[
     {
          "hostname" :  "x",
          "os" : "x"
          "ip" : "x.x.x.x"
          "group" : "x
     }
]
```

Vérifiable avec un « curl » (conseillé pour s'assurer des informations reçues)

```
curl -H 'X-Auth-Token:token-trouvé-dans-LibreNMS'http://127.0.0.1/api/v0/oxidized
```

Le technicien remplit la section source du fichier config en fonction du résultat du JSON. Ensuite, avec ces informations, Oxidized va tenter de se connecter. Pour cela, il va utiliser les informations de la section **input** mais aussi **username, password, timeout, retries** mais il va aussi utiliser une information essentielle **model_map**. L'OS du routeur donc le fichier « model » où il va trouver les commandes à effectuer. Une fois cela fait, il va stocker ces informations en fonction de la section **output**.

A noter que le username et le password doivent donner accès à la lecture en SSH ou autre moyen de connexion avec le routeur .

3. Comment le paramétrer :

- Configurer le fichier config

Attention très sensible l'indentation doit être parfaite.

```
# === Identifiants de connexion par défaut ===
username : admin                      # Nom d'utilisateur utilisé pour les connexions SSH
password : admin                      # Mot de passe par défaut


# === Intervalle de récupération ===
interval: 300                         # Fréquence de récupération (en secondes) — ici toutes les 5 minutes


# === Journalisation ===
use_syslog: false                     # Ne pas utiliser syslog (journal système)
log: /home/oxidized/.config/oxidized/logfile.log  # Fichier local de log
debug: true                           # Active les logs de débogage (utile pour diagnostiquer)


# === Paramètres d'exécution ===
threads: 30                           # Nombre de connexions simultanées
timeout: 20                           # Timeout pour chaque équipement (en secondes)
retries: 3                            # Nombre de tentatives de connexion avant abandon


# === Expression régulière pour reconnaître le prompt ===
prompt: !ruby/regexp /^"([\[\]A-Za-z0-9\(\)\@\/\.\-]+?)[\]>]\z/
# Sert à détecter la fin de la commande (prompt) pour chaque type d'équipement


# === Interface web / API REST d'Oxidized ===
rest: 0.0.0.0:8888                    # Serveur REST ouvert à toutes les interfaces, port 8888


# === Exécution immédiate d'un job si un équipement est ajouté ===
next_adds_job: true                   # Dès qu'un nouvel équipement est ajouté, il est interrogé immédiatement
```

```
# === Variables globales ===
vars:
  remove_secret: true              # Masque les secrets (mot de passe, clés) dans les config extraites


# === PID et gestion des crashs ===
pid: "/home/oxidized/.config/oxidized/pid"        # Fichier PID pour suivre le processus
crash:
  directory: "/home/oxidized/.config/oxidized/crashes"  # Emplacement des dumps en cas de crash


stats:
  history_size: 10                  # Nombre de versions conservées dans la mémoire (affichage rapide)


# === Méthode de connexion aux équipements ===
input:
  default: ssh                      # Connexion SSH par défaut
  ssh:
    secure: false


# === Stockage des configurations ===
output:
  default: git
  git:
    user: oxidized                  # Nom utilisé dans les commits
    email: git@isotelecom.com       # Email utilisé dans les commits
    repo: "/var/lib/oxidized/devices.git"  # Répertoire Git où les configurations sont stockées


# === Source externe des équipements (via API LibreNMS) ===
source:
  default: http
  http:
    url: "http://ip/api/v0/oxidized"  # URL de l’API qui retourne les équipements
    map:
```

```
      name: hostname              # Nom d'hôte utilisé comme identifiant
      model: os                   # Système d'exploitation de l'équipement
      ip: ip                      # Adresse IP de connexion
      group: group                # Groupe auquel appartient l'équipement
    headers:
      X-Auth-Token: token-LibreNMS # Token d'authentification
  reload: true                     # Recharge la source automatiquement à chaque
cycle


# === Modèles personnalisés pour certains OS ===
model_map:
  mikrotik: routeros               # Si model = mikrotik, utiliser le modèle
routeros
  routeros: routeros               # Alias explicite
  firewareos: firewareos           # Pour les équipements Firebox WatchGuard
  pfsense: pfsense                 # Pour les équipements pfSense


# === Groupes spécifiques d'équipements (il est possible aussi de mettre des mots
de passe différents pour chaque groupe) ===
groups:
  fireware:
    vars:
      ssh_port: 4118              # Port SSH personnalisé pour les équipements
fireware
  default:
    vars:
      ssh_port: 22                # Port SSH par défaut pour tous les autres
équipements
```

- Créer des groupes en fonction de l'os

Voici un exemple pour FirewareOS.

Dans le fichier :
/opt/librenms/config.php

Il faudra ajouter le code suivant :

```
#oxidized groupe par OS
$config['oxidized']['group_support'] = true;
```

```
$config['oxidized']['maps']['group']['os'][] = array('match' => 'fireware',
'group' => 'fireware');
```

- Donner l'accès à l'onglet Config aux utilisateurs en lecture

Dans le fichier :

/opt/librenms/app/Http/Controllers/Device/Tabs/ShowConfigController.php

Il faut retirer les lignes suivantes :

```
- public function visible(Device $device): bool
- {
-     return $this->oxidizedEnabled($device) || $this->findRancidConfigFile();
- }
```

Et les remplacer par celles-ci :

```
+ public function visible(Device $device): bool
+ {
+     return true;
+ }
```

4. Sources utiles :

Tutoriel plutôt complet : https://www.youtube.com/watch?v=KDzzW4ICv5A et https://www.youtube.com/watch?v=1H-konCmt0c

Documentation officielle LibreNMS : https://docs.librenms.org/Extensions/Oxidized/

Documentation officielle Oxidized : https://github.com/ytti/oxidized

Groupes : https://github.com/ytti/oxidized/issues/2586