TP : Surveillance à l'aide de ELK

Attention

La VM fournie pour le TP est configurée avec 2Go de RAM. Cela n'est pas suffisant pour faire fonctionner tous les composants du TP en même temps (*rsyslog*, *logstash*, *elasticsearch*, *kibana*). Il y a donc deux solutions :

- si votre ordinateur est assez puissant pour pouvoir ajouter de la RAM à la VM, la configurer avec 3 à 4 Go de RAM pour pouvoir tout faire fonctionner en parallèle ;
- si non allumez les services un par un. En général vous pouvez toujours laisser *rsyslog* et avoir soit *logstash* et *elasticsearch* en marche soit *elasticsearch* et *kibana*.

1 Prise en main

Dans cette première partie du TP, vous allez prendre en main la chaîne de traitement de logs : *rsyslog*, *logstash*, *elasticsearch* et enfin *kibana*.

Pour faciliter la prise en main, nous allons utiliser les logs locaux de la machine d'analyse qui seront envoyés dans la chaîne de traitement par l'instance locale de *rsyslog*.

Dans la suite du TP, nous allons envoyer des logs d'autres machines dans la chaîne de traitement à l'aide de *rsyslog* pour les machines Unix et pour le collecteur et *nxlog* pour les machines Windows.

1.1 Rsyslog

- Deux fichiers de configuration de *rsyslog* sont importants pour ce TP :
  - `/etc/rsyslog.d/30-logstash-json.conf` : permet de formatter les logs à envoyer dans un format JSON facilement exploitable ;
  - `/etc/rsyslog.d/40-forward-all-to-localhost.conf` : permet d'envoyer tous les logs vers une instance de *logstash* locale (configurée pour écouter sur le port 10514) ;
  - Cette configuration *rsyslog* a pour rôle d'envoyer les logs locaux dans la chaîne de traitement.

— Lancez le daemon `rsyslog` :

```
sudo systemctl start rsyslog
```

— Pour tester le bon fonctionnement de `rsyslog` regardez les paquets passer sur l'interface `lo` et faite une action qui génére un log (comme sudo ls). Vous devriez alors voir des requêtes ICMP vers `127.0.0.0:10514` (ce qui est normal vu que `logstash` n'est pas encore lancé).

1.2 Logstash

— Pour lancer *logstash*, vous utiliserez le fichier `logstash.conf` se trouvant dans le *home* de *user*. Il y a trois parties dans ce fichier :
  — la partie *input* indiquant d'attendre les logs sur le port 10514 au format JSON ;
  — la partie *filter* qui fait une manipulation du champ *host* ;
  — la partie *output* qui indique d'envoyer les logs traités vers l'instance locale d'*elasticsearch*.

— Lancez *logstash* (de préférence dans une session `tmux`) avec la commande suivante (*logstash* est correctement lancé quand "*Pipeline main started*" apparait) :

```
/opt/logstash/bin/logstash -f logstash.conf

Settings: Default pipeline workers: 1
Connection refused (Connection refused) {:class=>...
Pipeline main started
```

— Pour vérifier le bon fonctionnement de *logstash*, effectuez une action qui génère un log et vérifiez que vous obtenez quelque chose de la sorte (qui est le résultat normal car *elasticsearch* n'est pas lancé) :

```
Attempted to send a bulk request to Elasticsearch
configured at '["http://localhost:9200"]', but
Elasticsearch appears to be unreachable or down!
{:error_message=>"Connection refused (Connection
refused)", :class=>"Manticore::SocketException",
:level=>:error}
```

— À cette étape vous avez donc :
  — *rsyslog* qui envoie les logs locaux à *logstash* ;

— *logstash* effectue des traitements basiques sur les logs ;
— puis tente de les envoyer dans *elasticsearch*.

1.3 Elasticsearch

— Lancez *elasticsearch* :

```
sudo systemctl start elasticsearch
```

— Vérifiez que *elasticsearch* est bien lancé avec la requête suivante (permettant de récupéré l'état de indices *elasticsearch*) :

```
curl 'http://localhost:9200/_cat/indices?pretty'
```

— À cette étape vous avez :
  — *rsyslog* qui envoie les logs locaux à *logstash* ;
  — *logstash* qui effectue des traitements basiques sur les logs ;
  — *logstash* qui envoie les logs dans *elasticsearch*.
— Pour vérifier que tout fonctionne, effectuez une action générant des logs locaux et vérifiez qu'un index a bien été créé.

1.4 Kibana

— Si vous n'avez pas ajouté de RAM à votre VM, il faudra éteindre *logstash* avant de lancer *kibana*.
— Lancez `kibana` :

```
sudo systemctl start kibana
```

— Pour tester `kibana`, vous avez deux options :
  — la première, plus couteuse en RAM pour la VM, consiste, depuis l'interface de VirtualBox, à démarrez le serveur X11 (startx) de la VM. Lancez `firefox` et naviguez vers `127.0.0.1:5601` ;
  — la deuxième solution consiste à modifier le fichier de configuration de *kibana* (`/etc/kibana/kibana.yml`) en mettant l'IP de la VM sur le réseau de connexion avec la machine hôte dans le paramètre *server.host*. Pour accèder à *kibana*, naviguez depuis un navigateur sur la machine hôte vers `<ipdelaVM>:5601`.
— Vous visualisez alors l'intreface de `kibana`. Allez dans l'onglet *Discover*. Vous devriez alors voir les logs que vous avez généré précédemment.

1.5 Premère utilisation

— Le but de cette partie va être de visualiser les utilisateurs générant des logs sur votre VM.
— Pour ce faire, vous allez :
  — Écrire la configuration *logstash* permettant d'extraire le nom d'utilisateur d'au moins un type de log ;

Solution
On pourrait choisir par exemple de *parser* les logs de connexion SSH réussie. Pour faire ça on ajoute dans la partie *filter* du fichier de configuration de *logstash* :

```
  grok {
    match => {
      "message" => "Accepted password for
%{DATA:user} from %{IP} port %{INT} ssh2"
    }
  }
```

  — Vérifier le bon fonctionnement de votre configuration *logstash* en effectuant des requêtes *elasticsearch* en python ;

Solution
En se connectant en SSH sur la machine, on génère un log qui correspond à celui que l'on veut *parser*.
On peut alors requêter *elasticsearch* en python pour voir si notre grok fonctionne :

```
from elasticsearch import Elasticsearch
es = Elasticsearch()

res = es.search(index="logstash-*",
                body={"query":
                      { "exists": {
                        "field": "user"
                      }}})
```

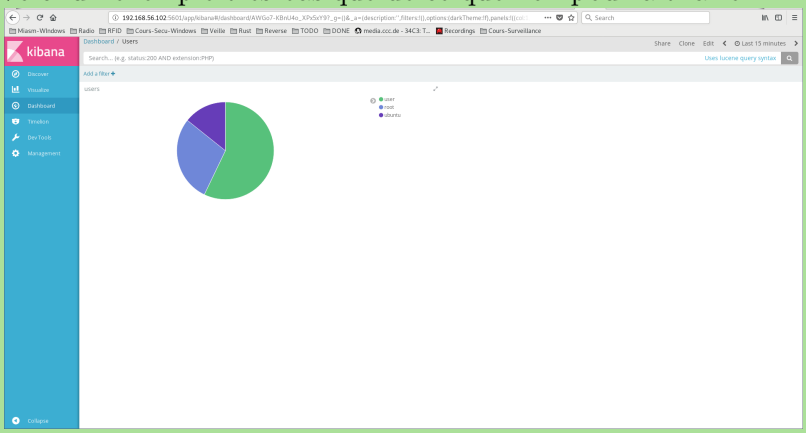
— Visualer le résultat dans *kibana* puis créer un *dashboard* permettant de visualiser simplement les principaux utilisateurs de votre VM.

Tips

Pour pouvoir manipuler votre champ contenant le nom de l'utilisateur, il peut être nécessaire de rafraîchir les champs connus de *kibana* (Management->Index Patterns->Refresh field list).

Solution

Voici un exemple très basique de ce que l'on pourrait faire :



2 Application

Dans cette partie, vous allez mettre en place des moyens de détection et de visualisation des attaques que vous avez réalisé dans le TP d'hier.

2.1 Collecte

La première étape va être de récupérer les logs contenant les traces de vos attaques. Ces logs sont normalement stockés sur votre machine de collecte (*vmcollecte*). Transferrez les sur la machine d'analyse de la manière de votre choix.

Dans un environnement d'entreprise le transfert des logs entre la machine de collecte et celle d'analyse se ferait grâce à l'instance *rsyslog* de la machine de collecte qui enverrait les logs :

— soit directement aux instances *logstash* de la machine d'analyse ;
— soit à des instances d'outil comme *kafka* permettant de faire de la retention.

2.2 Extraction et indexation

— Pour envoyer nos logs dans *logstash* (qui les insérera dans *elasticsearch*) nous allons simplement utiliser *nc*.

```
cat fichier.log | nc localhost 10514
```

— Dans la configuration actuelle, *logstash* attend des logs au format JSON. En faisant du *cat/nc*, on envoie des lignes à *logstash*. Voici la configuration permettant de prendre cela en compte :

```
input {
  tcp {
    port => 10514
    codec => "line"
  }
}
```

— Appliquez cette nouvelle configuration et envoyez les logs. Que remarquez vous ?

Solution
Le champ *message* contient maintenant tout le log. Certaines informations intéressantes ne sont donc pas extraites comme le nom de la machine ou encore le nom du programme ayant généré le log.

— Modifiez la configuration *logstash* pour prendre cela en compte.

Solution
Dans la configuration *logstash*, on aurait pu rajouter cette configuration pour extraire les informations souhaitées :

```
 grok {
    match => {
      "message" =>
"%{SYSLOGTIMESTAMP:syslog_timestamp}
 %{DATA:syslog_hostname}
```

```
  %{WORD:programname}%{DATA}:
  %{GREEDYDATA:syslog_message}"
      }
    }

    grok {
      match => {
        "syslog_message" => "Accepted password for
%{DATA:user} from %{IP:ip} port %{INT:port} ssh2"
      }
    }
```

Tips

Pour faire vos tests, vous aurez peut être besoin de nettoyer *elasticsearch*. Vous pouvez faire cela en supprimant les indexes de votre choix. Par exemple en python :

```
from elasticsearch import Elasticsearch
es = Elasticsearch()

es.indices.delete(index='logstash-2019.02.26',
                  ignore=[400, 404])
```

2.3 Analyse

— Reprenez chaque étape de votre attaque lors du TP d'hier et mettez en place un moyen de la détecter :
  — en extrayant les informations utiles des logs ;
  — en forgeant une requête *elasticsearch* mettant en valeur votre attaque ;
  — en créant un élément *kibana* permettant de visualiser l'attaque.

Attention

Il vous manque peut être de l'information pour détecter certaines étapes. Il

faudrait soit avoir les captures réseau (voir le cours de demain) soit récolter plus d'information au niveau système (utilisation d'*auditd*) et applicatif.

3 Logs Windows

Le but de cette partie est de mettre en place un moyen de détection d'utilisations de l'outil Mimikatz.

Comme machine Windows générant les logs nous allons utiliser la station Windows 7 du cours de sécurité Windows.

3.1 NXlog

— Téléchargez NXlog depuis `https://nxlog.co/products/nxlog-community-edition/download` ;
— Installez NXlog sur la station Windows ;

```
msiexec /i nxlog-ce-<VERSION>.msi
```

— NXlog s'est installé sous forme de service Windows. Vérifiez que le service existe.
— Ajoutez la configuration suivante (dans `c:\ProgramFiles(x86)\nxlog\conf\nxlog.conf`)

```
Panic Soft
#NoFreeOnExit TRUE

define ROOT     C:\Program Files (x86)\nxlog
define CERTDIR  %ROOT%\cert
define CONFDIR  %ROOT%\conf
define LOGDIR   %ROOT%\data
define LOGFILE  %LOGDIR%\nxlog.log
LogFile %LOGFILE%

Moduledir %ROOT%\modules
CacheDir  %ROOT%\data
Pidfile   %ROOT%\data\nxlog.pid
SpoolDir  %ROOT%\data

<Extension json>
    Module                  xm_json
```

```
</Extension>

<Input eventlog>
    Module      im_msvistalog
</Input>

<Output logstash>
    Module      om_udp
    Host        <IP MACHINE ANALYSE>
    Port        514
    Exec        to_json();
</Output>

<Route 1>
    Path        eventlog => logstash
</Route>
```

— Comme IP de collecteur, mettez l'IP de votre machine d'analyse. Pour éviter d'avoir besoin de trop de VM en même temps nous allons directement envoyer les logs dans une instance *logstash* de notre machine d'analyse plutôt qu'à une instance *rsyslog* de la machine de collecte ;
— Redémarrez NXlog (si le service ne démarre pas, il y a des logs dans `c:\ProgramFiles(x86)\nxlog\data\nxlog.log`) ;
— Vérifiez que les logs sont bien envoyés au collecteur (à l'aide de *tcpdump* par exemple).

3.2 Logstash

— Nous allons modifier la configuration de *logstash* pour pouvoir récupérer les logs provenant de NXlog :

```
input {
  udp {
    port => 514
    codec => "json"
  }
}
```

— Démarrez *logstash* avec cette configuration et vérifiez qu'ils arrivent bien dans *elasticsearch*.

3.3 Sysmon

— Pour pouvoir détecter une utilisation de Mimikatz (`sekurlsa::logonpasswords`), il faut chercher les processus tentant d'obtenir des droits de lecture sur quel processus ?

Solution
Pour récupérer les secrets d'authentification, Mimikatz effectue un accès en lecture dans la mémoire du processus *lsass* (*cf.* de cours sécurité Windows).

— Pour obtenir les logs d'accès à la mémoire des processus, il faut demander à Windows de nous fournir plus de logs, nous allons alors utiliser l'outil *Sysmon* de *Sysinternals* ;
— Téléchargez la dernière version de *Sysmon* sur `https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon` ;
— Voici le fichier de configuration *Sysmon* permettant de générer des logs pour les tentatives d'accès à la mémoire d'un processus :

```
<Sysmon schemaversion="4.22">
    <EventFiltering>
        <RuleGroup name="" groupRelation="or">
            <ProcessAccess onmatch="include">
                <TargetImage condition="is">
                CHEMIN_DU_PROCESS</TargetImage>
            </ProcessAccess>
        </RuleGroup>
    </EventFiltering>
</Sysmon>
```

— Pour installer *Sysmon* :

```
sysmon.exe -i sysmonconfig.xml
```

— Pour recharger la configuration de *Sysmon* :

```
sysmon.exe -c sysmonconfig.xml
```

— Redémarrez *NXlog* ;
— Lancez Mimikatz et récupérez les *hashs* stockés en mémoire. Quels logs obtenez-vous ?

Solution

Nous obtenons des logs de la *Category* "Process accessed (rule : ProcessAccess)". Ces logs possèdent un champ *GrantedAccess* correspondant au type d'accès demandé sur la mémoire du processus.

Le processus faisant l'accès est indiqué dans le champ *SourceImage*.

Le process accèdé est indiqué dans le champ *TargetImage*

Lorsque que nous lançons *sekurlsa ::logonpasswords*, on voit que Mimikatz demande les droits d'accès 0x1010.

3.4 Détection

— À l'aide des informations que trouverez sur `https://docs.microsoft.com/en-us/windows/desktop/ProcThread/process-security-and-access-rights`, expliquez la valeur que vous obtenez dans le champ GrantedAccess quand vous lancez Mimikatz ;

Solution

Lorsque que nous lançons *sekurlsa ::logonpasswords*, on voit que Mimikatz demande les droits d'accès 0x1010.

Or le droit d'accès 0x0010 correspond à `PROCESS_VM_READ` permettant de lire la mémoire d'un processus.

— Modifiez la configuration *logstash* pour ajouter un *tag* permettant d'identifier les tentatives d'accès en lecture à *lsass*. Ce tag permettera d'identifier Mimikatz et plus largement tous les processus capables de récupérer les secrets d'authentification stockés dans la mémoire de *lsass* ;

Solution

```
filter {
  if [GrantedAccess] == "0x1010" and [TargetImage] ==
"C:\Windows\system32\lsass.exe" {
      mutate {
        add_tag => [ "mimikatz" ]
      }
  }
```

```
  # Plus générique
  if [GrantedAccess] =~
"^0x[a-f0-9]*[13579bdf][a-f0-9]$" and [TargetImage]
== "C:\Windows\system32\lsass.exe" {
      mutate {
        add_tag => [ "mimikatz_like" ]
      }
  }
}
```

— Grâce à ce tag vous pourrez facilement retrouver les événements suspects pour lever des alertes depuis *logstash* ou pour les afficher dans *kibana*.