Simulation des attaques

SCÉNARIO 1 — Scan réseau :

But : détecter qu'un hôte scanne des ports (reconnaissance préliminaire).
Justification : Les scans sont souvent la première étape d'une attaque (reconnaissance). La détecter tôt permet de réduire la surface d'attaque et d'investiguer sur une machine suspecte.

La commande à exécuter sur la machine attaquante :

```
nmap -sS -p 1-1024 IP_VM_SNORT
```

La règle snort qui détecte cette attaque :

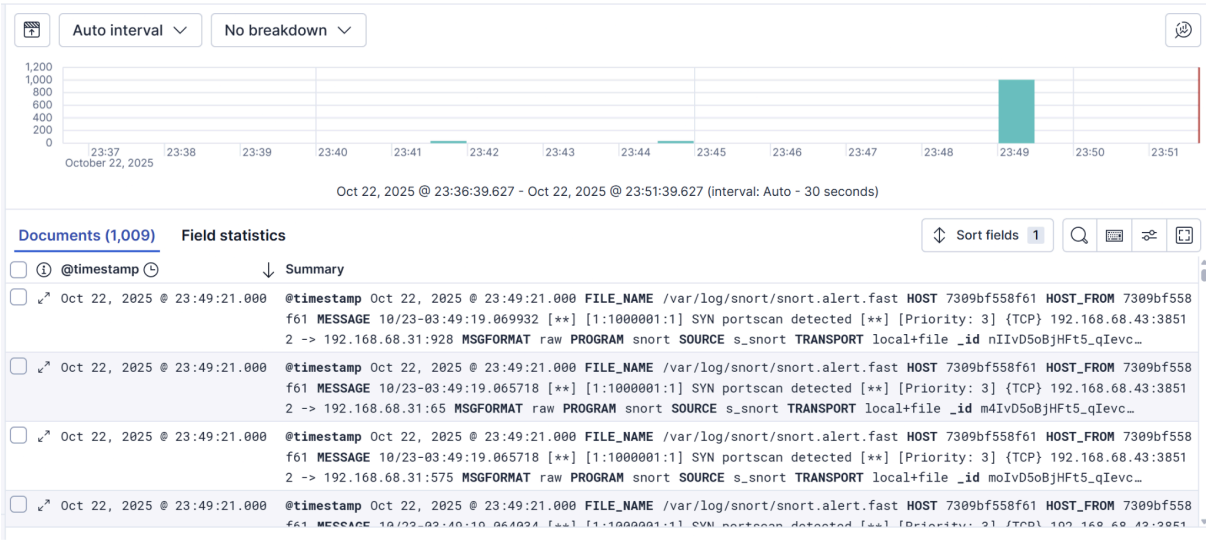
```
# detect multiple SYNs from same source to many dst ports
alert tcp any any -> any any (msg:"SYN portscan detected"; flags:S;
detection_filter: track by_src, count 20, seconds 10; priority:3;
sid:1000001; rev:1;)
```

On voit que snort détecte l'attaque :

```
10/23-03:49:19.067100  [**] [1:1000001:1] SYN portscan detected [**]
[Priority: 3] {TCP} 192.168.68.43:38512 -> 192.168.68.31:428
10/23-03:49:19.067100  [**] [1:1000001:1] SYN portscan detected [**]
[Priority: 3] {TCP} 192.168.68.43:38512 -> 192.168.68.31:842
10/23-03:49:19.069932  [**] [1:1000001:1] SYN portscan detected [**]
[Priority: 3] {TCP} 192.168.68.43:38512 -> 192.168.68.31:928
10/23-03:49:19.069932  [**] [1:1000001:1] SYN portscan detected [**]
[Priority: 3] {TCP} 192.168.68.43:38512 -> 192.168.68.31:498
10/23-03:49:19.069933  [**] [1:1000001:1] SYN portscan detected [**]
[Priority: 3] {TCP} 192.168.68.43:38512 -> 192.168.68.31:452
10/23-03:49:19.069933  [**] [1:1000001:1] SYN portscan detected [**]
[Priority: 3] {TCP} 192.168.68.43:38512 -> 192.168.68.31:42
```

(sortie du fichier /var/log/snort/snort.alert.fast)

On retrouve ces alertes dans Kibana :



Voila un exemple de log de l'alerte : {


```
{
  "@timestamp": [
    "2025-10-23T03:49:21.000Z"
  ],
  "FILE_NAME": [
    "/var/log/snort/snort.alert.fast"
  ],
  "FILE_NAME.keyword": [
    "/var/log/snort/snort.alert.fast"
  ],
  "HOST": [
    "7309bf558f61"
  ],
  "HOST.keyword": [
    "7309bf558f61"
  ],
  "HOST_FROM": [
    "7309bf558f61"
  ],
  "HOST_FROM.keyword": [
    "7309bf558f61"
  ],
  "MESSAGE": [
    "10/23-03:49:19.069932  [**] [1:1000001:1] SYN portscan detected [**]
[Priority: 3] {TCP} 192.168.68.43:38512 -> 192.168.68.31:928"
  ],
  "MESSAGE.keyword": [
    "10/23-03:49:19.069932  [**] [1:1000001:1] SYN portscan detected [**]
[Priority: 3] {TCP} 192.168.68.43:38512 -> 192.168.68.31:928"
  ],
  "MSGFORMAT": [
    "raw"
  ],
```

```
  "MSGFORMAT.keyword": [
    "raw"
  ],
  "PROGRAM": [
    "snort"
  ],
  "PROGRAM.keyword": [
    "snort"
  ],
  "SOURCE": [
    "s_snort"
  ],
  "SOURCE.keyword": [
    "s_snort"
  ],
  "TRANSPORT": [
    "local+file"
  ],
  "TRANSPORT.keyword": [
    "local+file"
  ],
  "_id": "nIIvD5oBjHFt5_qIevcZ",
  "_index": "logs-2025.10.23",
  "_score": null
}
```


SCÉNARIO 2 — Brute-force SSH

But : détecter tentatives répétées de login SSH (probable brute force).
Justification : En cas d'accès SSH compromis, l'impact peut être élevé. Il est important de prioriser ces alertes, de la détecter et de la bloquer.

Les commandes à exécuter sur la machine attaquante :

```
echo -e "password\n123456\nroot" > /home/user/pwlist.txt
hydra -l root -P /home/user/pwlist.txt ssh://IP_VM_SNORT
```

La règle snort qui le détecte est :

```
alert tcp any any -> any 22 (msg:"SSH brute force"; flow:to_server,established;
detection_filter: track by_src, count 5, seconds 60; priority:1; sid:1000002;
rev:1;)
```

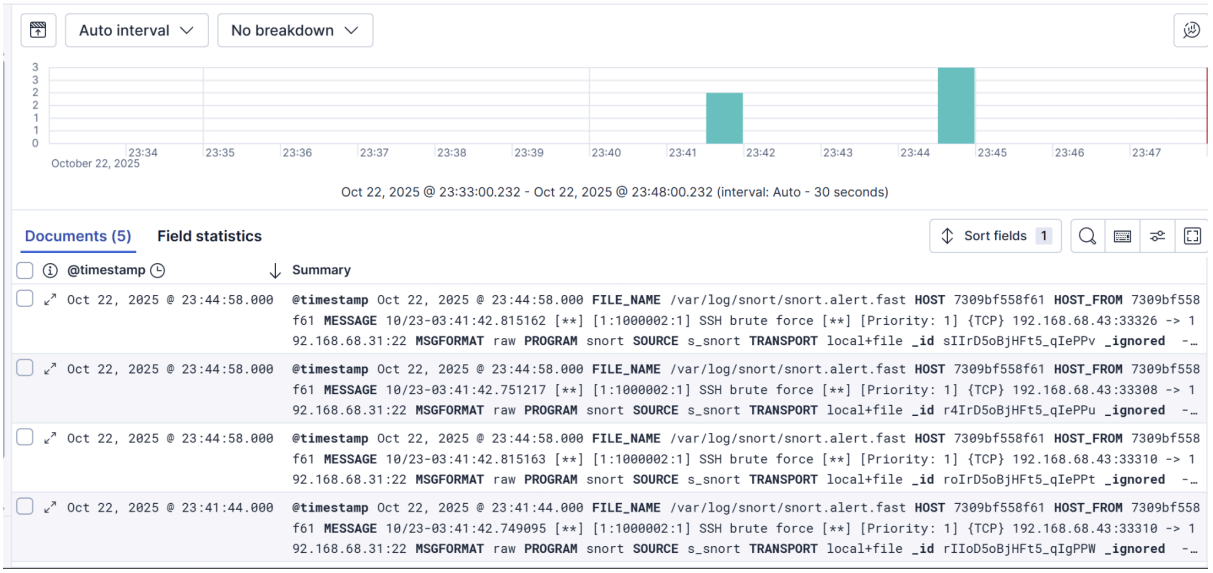
Snort détecte bien l'attaque :

```
10/23-03:41:42.815163  [**] [1:1000002:1] SSH brute force [**] [Priority: 1]
```

```
{TCP} 192.168.68.43:33310 -> 192.168.68.31:22
10/23-03:41:42.815162  [**] [1:1000002:1] SSH brute force [**] [Priority: 1]
{TCP} 192.168.68.43:33326 -> 192.168.68.31:22
10/23-03:41:42.751217  [**] [1:1000002:1] SSH brute force [**] [Priority: 1]
{TCP} 192.168.68.43:33308 -> 192.168.68.31:22
```

(sortie du fichier /var/log/snort/snort.alert.fast)

On voit la remonté des alertes dans kibana :



Le message du log : [

```
      "10/23-03:41:42.815162  [**] [1:1000002:1] SSH brute force [**] [Priority: 1]
{TCP} 192.168.68.43:33326 -> 192.168.68.31:22"
```

SCÉNARIO 3 — Tentative d’exploitation Web (PHP exploit / param exploit=1) :

But : détecter requête HTTP contenant payload d'exploitation (ex: `?exploit=1`).
Justification : attaque applicative courante qui mène souvent à RCE.

La commande a faire sur la machine attaquante :

```
curl "http://IP_VM_SNORT/index.php?exploit=1"
```

La règle qui détecte l’attaque :

```
# 3 - Requête HTTP suspecte (exploit PHP)
alert tcp any any -> any 80 (msg:"HTTP exploit attempt";
```

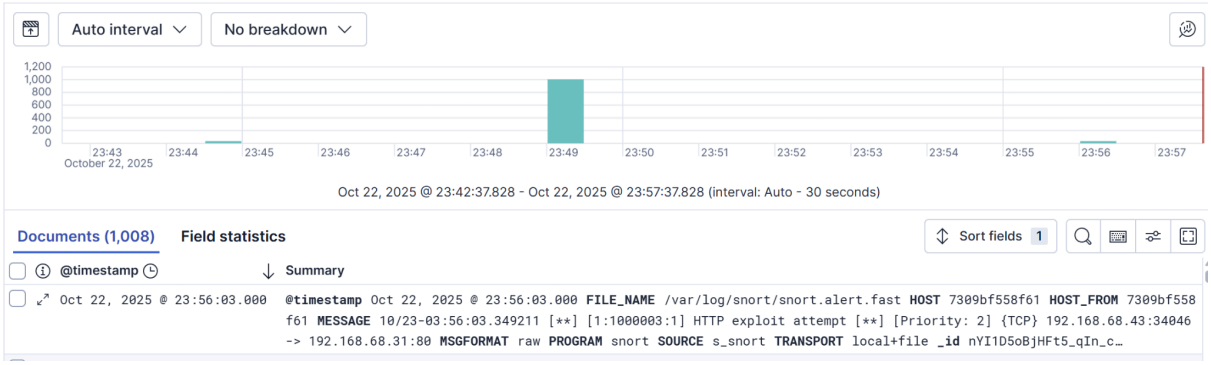
```
content:"exploit"; http_uri; nocase; priority:2; sid:1000003; rev:1;)
```

Snort a levé une alerte :

```
10/23-03:56:03.349211  [**] [1:1000003:1] HTTP exploit attempt [**]
[Priority: 2] {TCP} 192.168.68.43:34046 -> 192.168.68.31:80
```

(sortie du fichier /var/log/snort/snort.alert.fast)

On retrouve notre alerte dans Kibana :



Le contenu du message en alerte est :

```
"10/23-03:56:03.349211  [**] [1:1000003:1] HTTP exploit attempt [**] [Priority: 2]

{TCP} 192.168.68.43:34046 -> 192.168.68.31:80"
```

SCÉNARIO 4 — Requête HTTP suspicious .exe

But : détecter téléchargement/exfiltration de binaires .exe via HTTP (ou tentative d'accès).
Justification : Si nous détectons un téléchargement exécutable, il y a un risque qu'il s'agisse d'un malware.
En réalité, la règle ne sera sûrement pas celle-ci, nous ne signalerons pas tous les téléchargement d'exe. Il faudrait affiner la règle pour détecter les exe potentiellement malveillants avec des listes de malwares connus ou autre. Nous avons fait une règle un peu large pour faciliter son implémentation dans le cadre du projet.

La commande sur la machine attaquante :

```
for i in $(seq 1 5); do curl -s -o /dev/null
"http://IP_VM_SNORT/malware.exe"; sleep 0.2; done
```

La règle snort qui détecte le téléchargement est :

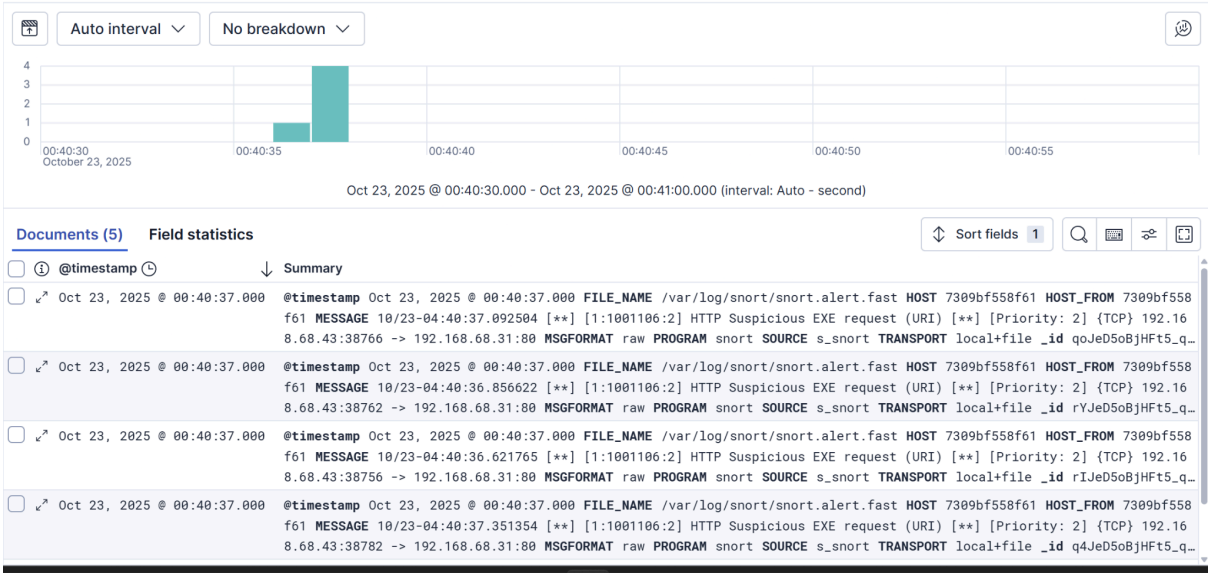
```
# 4 - Téléchargement exécutable suspect (.exe)
```

```
alert tcp any any -> $HOME_NET $HTTP_PORTS \
 (msg:"HTTP Suspicious EXE request (URI)"; flow:to_server;
uricontent:".exe"; nocase; priority:2; sid:1001106; rev:2;)
```

Snort détecte le téléchargement de l'exécutable.

```
10/23-04:40:36.621765  [**] [1:1001106:2] HTTP Suspicious EXE request
(URI) [**] [Priority: 2] {TCP} 192.168.68.43:38756 -> 192.168.68.31:80
10/23-04:40:36.856622  [**] [1:1001106:2] HTTP Suspicious EXE request
(URI) [**] [Priority: 2] {TCP} 192.168.68.43:38762 -> 192.168.68.31:80
10/23-04:40:37.092504  [**] [1:1001106:2] HTTP Suspicious EXE request
(URI) [**] [Priority: 2] {TCP} 192.168.68.43:38766 -> 192.168.68.31:80
10/23-04:40:37.351354  [**] [1:1001106:2] HTTP Suspicious EXE request
(URI) [**] [Priority: 2] {TCP} 192.168.68.43:38782 -> 192.168.68.31:80
```

On retrouve nos alertes dans kibana :



Le message remonté par l'alerte est :

```
"10/23-04:40:37.092504  [**] [1:1001106:2] HTTP Suspicious EXE request (URI) [**]

[Priority: 2] {TCP} 192.168.68.43:38766 -> 192.168.68.31:80"
```

SCÉNARIO 5 — ICMP flood

But : détecter un afflux massif de paquets ICMP depuis une même source (DoS).
Justification : ICMP flood altère la disponibilité — besoin de réponse réseau rapide.

La commande a mettre pour flood la vm est :

```
for i in $(seq 1 25); do sudo ping -c1 -W1 IP_VM_SNORT & done; wait
```

La règle pour détecter est :

```
# 5 - ICMP flood (DoS)
alert icmp any any -> any any (msg:"ICMP flood detected";
detection_filter: track by_src, count 20, seconds 10; priority:1;
sid:1000005; rev:1;)
```

Snort détecte bien le ICMP flood.

```
10/23-04:11:58.991897  [**] [1:1000005:1] ICMP flood detected [**]
[Priority: 1] {ICMP} 192.168.68.43 -> 192.168.68.31
10/23-04:11:58.991968  [**] [1:1000005:1] ICMP flood detected [**]
[Priority: 1] {ICMP} 192.168.68.31 -> 192.168.68.43
10/23-04:11:59.403896  [**] [1:1000005:1] ICMP flood detected [**]
[Priority: 1] {ICMP} 192.168.68.43 -> 192.168.68.31
10/23-04:11:59.403965  [**] [1:1000005:1] ICMP flood detected [**]
[Priority: 1] {ICMP} 192.168.68.31 -> 192.168.68.43
10/23-04:11:59.774632  [**] [1:1000005:1] ICMP flood detected [**]
[Priority: 1] {ICMP} 192.168.68.43 -> 192.168.68.31
10/23-04:11:59.774696  [**] [1:1000005:1] ICMP flood detected [**]
[Priority: 1] {ICMP} 192.168.68.31 -> 192.168.68.43
```

(sortie du fichier /var/log/snort/snort.alert.fast)

On retrouve bien nos “dos” dans kibana :

Le message dans l’alerte est :

```
"10/23-04:11:59.403896  [**] [1:1000005:1] ICMP flood detected [**] [Priority: 1]

{ICMP} 192.168.68.43 -> 192.168.68.31"
```