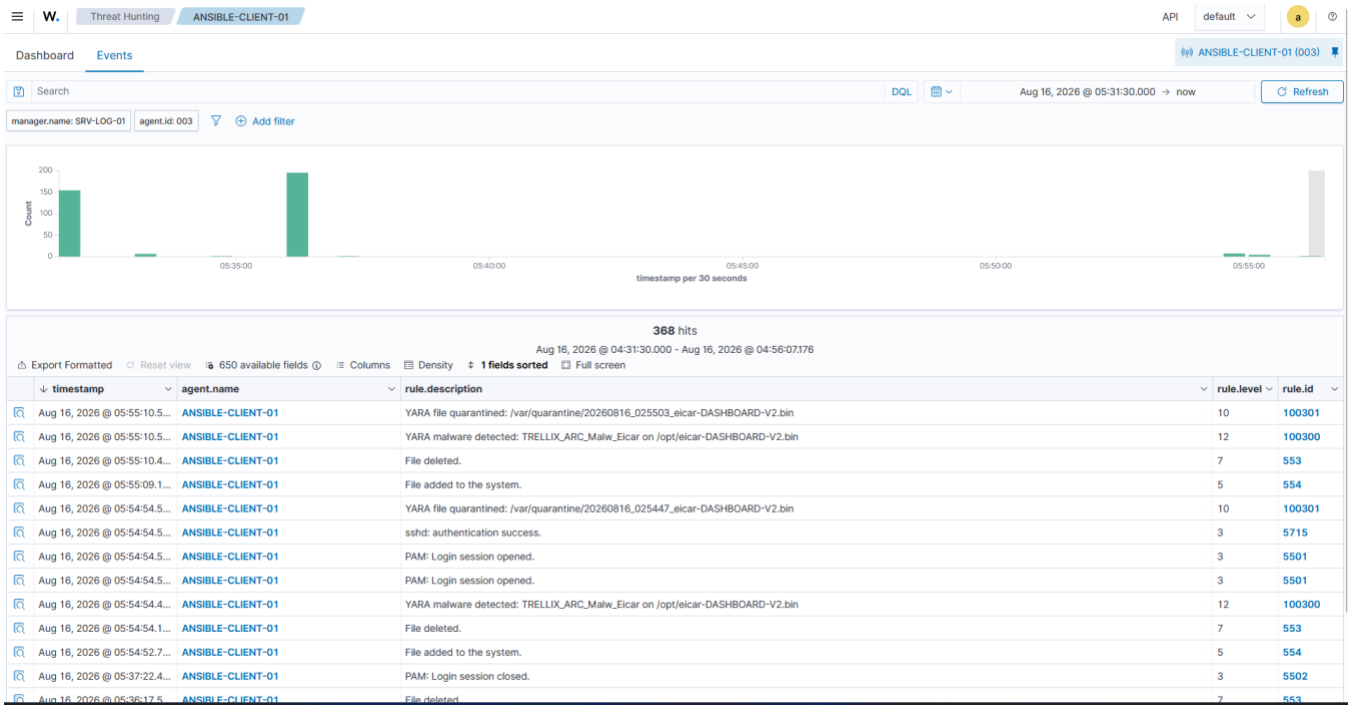
WAZUH + YARA

Active Response : détection et quarantaine automatique de fichiers malveillants

Tutoriel technique — Debian / Wazuh 4.14.x / Ansible



Résultat final : détection YARA (100300) puis confirmation de quarantaine (100301) dans Wazuh.

Réalisé par Adel Sadek

Août 2026

1. Objectif du tutoriel

Cette documentation décrit la brique de sécurité construite autour de Wazuh FIM, YARA Forge et Active Response. L'objectif est qu'un fichier créé ou modifié dans un répertoire surveillé soit analysé automatiquement par YARA. Si une règle YARA correspond, le fichier est déplacé dans une zone de quarantaine, ses droits sont retirés et deux alertes dédiées sont visibles dans le dashboard Wazuh.

But pratique — Pouvoir reconstruire la configuration à partir de zéro sans dépendre de ma mémoire : prérequis, fichiers, commandes, tests, erreurs rencontrées et corrections.

2. Ce qu'il faut comprendre avant de configurer

2.1 Wazuh FIM

FIM signifie File Integrity Monitoring. Wazuh-syscheckd surveille les chemins déclarés et produit des événements lorsqu'un fichier est ajouté, modifié ou supprimé. FIM ne décide pas qu'un fichier est un virus : il constate un changement.

2.2 YARA

YARA est un moteur de correspondance de règles. Une règle décrit des chaînes, motifs ou caractéristiques associées à une famille de malware ou à un comportement. YARA reçoit un fichier et un ruleset, puis renvoie le nom des règles qui correspondent.

2.3 YARA Forge Core

Au lieu d'écrire manuellement des centaines de règles, le projet utilise le paquet Core de YARA Forge. Dans notre installation, l'archive fournit un ruleset compilé sous la forme d'un grand fichier .yar : /opt/yara/packages/core/yara-rules-core.yar. Le fait de n'avoir qu'un fichier .yar est donc normal.

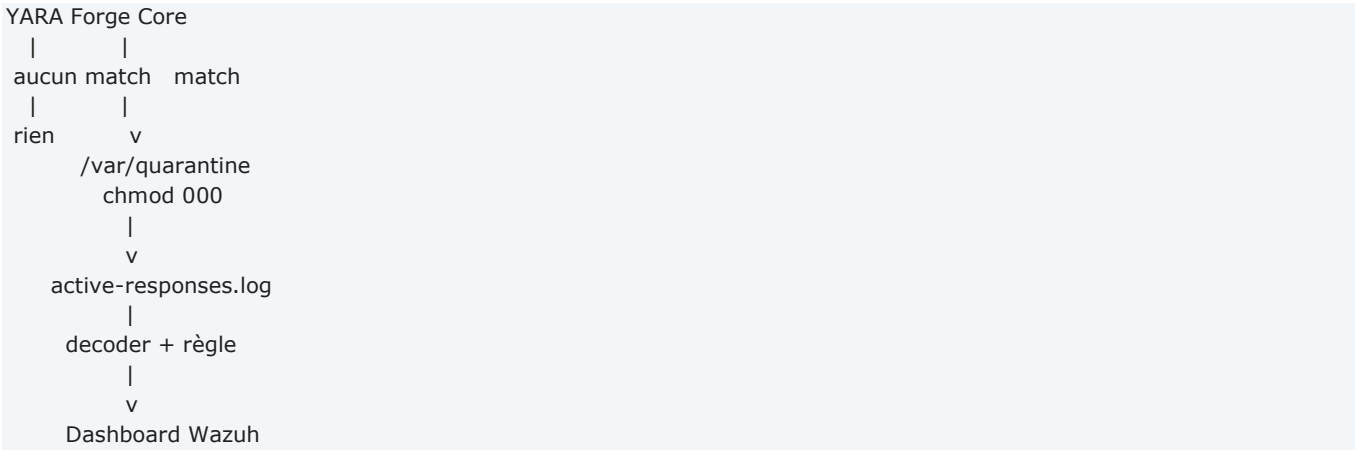
2.4 Active Response

Active Response permet à Wazuh d'exécuter une action lorsqu'une règle donnée se déclenche. Ici, les événements FIM de création/modification déclenchent yara.sh sur l'agent concerné. Le script récupère le chemin du fichier dans le JSON reçu sur l'entrée standard, lance YARA et ne met en quarantaine que si le scan réussit ET produit un vrai match.

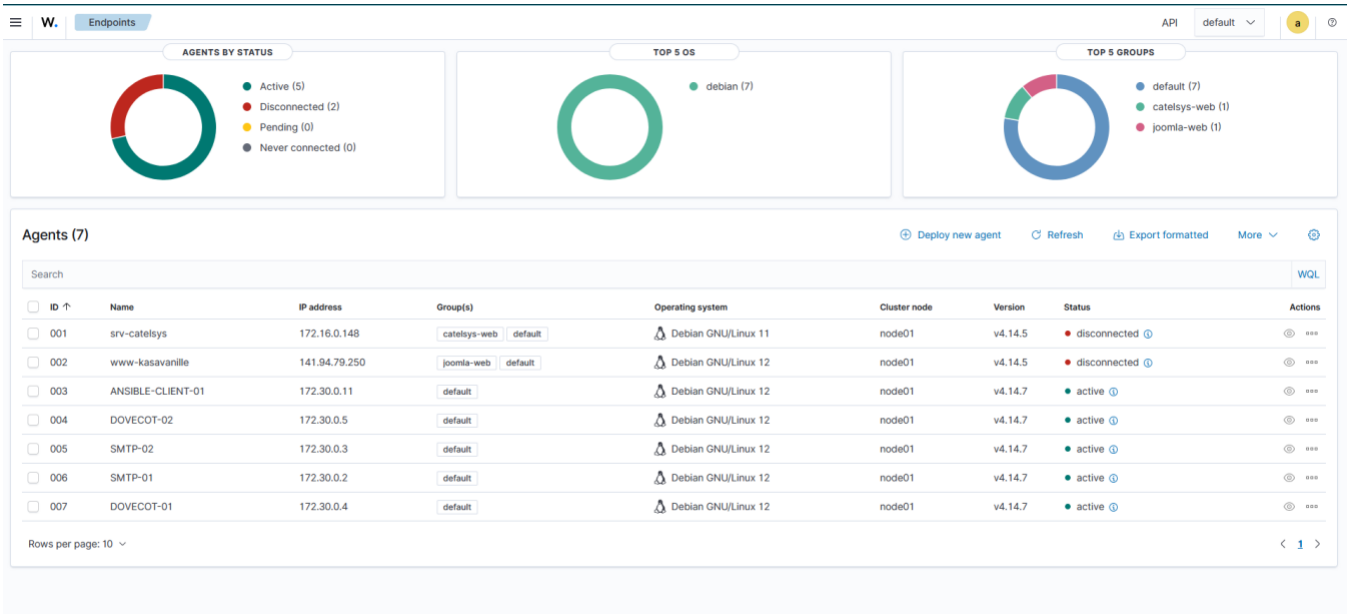
Point essentiel — FIM = détecte le changement. YARA = qualifie le fichier. Active Response = exécute l'action. La quarantaine = mesure de confinement.

2.5 Chaîne complète

```
Fichier créé ou modifié
        |
        v
Wazuh FIM / wazuh-syscheckd
        |
        v
Règle FIM 550 / 554
        |
        v
Active Response -> yara.sh
        |
        v
```


YARA Forge Core
aucun match
match
rien
/var/quarantine
chmod 000
active-responses.log
decoder + règle
Dashboard Wazuh

3. Architecture et prérequis



Exemple de flotte Wazuh : plusieurs agents Debian rattachés au manager.

Composants utilisés dans le laboratoire :

- Un Wazuh Manager : SRV-LOG-01.
- Un serveur Ansible : ANSIBLE-SRV-01.
- Un agent pilote : ANSIBLE-CLIENT-01.
- Wazuh Agent 4.14.x.
- YARA 4.2.3 sur le client pilote.
- YARA Forge Core.
- jq, utilisé par yara.sh pour lire le JSON transmis par Wazuh.
- Un répertoire de quarantaine /var/quarantine.

Méthode — Toujours valider sur un seul client avec --limit avant de généraliser à la flotte.

4. Installation de YARA avec Ansible

Le rôle Ansible crée les dépendances et les répertoires nécessaires.

```
- name: Installer YARA et les outils utiles
  ansible.builtin.apt:
    name:
      - yara
      - unzip
      - jq
    state: present
    update_cache: true

- name: Creer le repertoire YARA
  ansible.builtin.file:
    path: /opt/yara
    state: directory
    owner: root
    group: root
```

```
    mode: "0755"

- name: Creer le repertoire de quarantaine
  ansible.builtin.file:
    path: /var/quarantine
    state: directory
    owner: root
    group: root
    mode: "0700"
```

Vérifications :

```
ansible client01 -m shell -a "yara --version"
ansible client01 -m shell -a "ls -ld /opt/yara /var/quarantine"
```

Résultat attendu : YARA répond avec sa version et /var/quarantine est accessible uniquement à root au niveau du répertoire.

5. Déployer YARA Forge Core

La release Core a été choisie pour limiter l'empreinte disque tout en conservant un corpus de règles sérieux.

```
curl -s https://api.github.com/repos/YARAHQ/yara-forge/releases/latest | grep browser_download_url
```

Dans le rôle, télécharger l'archive Core puis l'extraire dans /opt/yara. Le résultat utile observé était :

```
/opt/yara/yara-forge-rules-core.zip
/opt/yara/packages/core/yara-rules-core.yar
```

À retenir — Un seul fichier yara-rules-core.yar peut contenir un très grand nombre de règles. 'wc -l' sur les fichiers .yar ne mesure donc pas le nombre de signatures.

Test simple sur un binaire sain :

```
yara -w /opt/yara/packages/core/yara-rules-core.yar /bin/ls
```

Aucune sortie signifie ici : aucune règle n'a correspondu.

6. Configurer Wazuh FIM

La première tentative consistait à surveiller '/', mais elle était trop large et difficile à exploiter. La configuration a été corrigée pour surveiller explicitement les emplacements importants en temps réel et laisser /boot en scan planifié.

```
<syscheck>
  <disabled>no</disabled>

  <directories realtime="yes">/bin</directories>
  <directories>/boot</directories>
  <directories realtime="yes">/etc</directories>
  <directories realtime="yes">/home</directories>
  <directories realtime="yes">/opt</directories>
  <directories realtime="yes">/root</directories>
  <directories realtime="yes">/sbin</directories>
  <directories realtime="yes">/tmp</directories>
  <directories realtime="yes">/usr/bin</directories>
  <directories realtime="yes">/usr/sbin</directories>
  <directories realtime="yes">/var/tmp</directories>
  <directories realtime="yes">/var/www</directories>

  <alert_new_files>yes</alert_new_files>

  <ignore>/run</ignore>
  <ignore>/var/quarantine</ignore>
  <ignore>/var/log</ignore>
  <ignore>/var/cache</ignore>
  <ignore>/var/lib/apt</ignore>
  <ignore>/var/lib/dpkg</ignore>
  <ignore>/root/.ansible/tmp</ignore>

  <ignore type="sregex">^/tmp/ansible_</ignore>
  <ignore type="sregex">^/tmp/apt\.</ignore>
  <ignore type="sregex">^/tmp/apt-key-gpghome\.</ignore>
  <ignore type="sregex">.log$|.swp$</ignore>

  <process_priority>10</process_priority>
  <max_eps>50</max_eps>
</syscheck>
```

Pourquoi exclure la quarantaine ? — Sans <ignore>/var/quarantine</ignore>, le déplacement d'un malware vers la quarantaine peut générer de nouveaux événements FIM et créer une boucle ou du bruit.

Pourquoi exclure Ansible/APT ? — Les modules Ansible et APT créent beaucoup de fichiers temporaires qui disparaissent immédiatement. YARA arrivait parfois après leur suppression, d'où les messages 'could not open file'.

6.1 Vérifier que le temps réel est réellement actif

```
ansible client01 -m shell -a "grep -iE 'syscheck|scan|realtime|inotify' /var/ossec/logs/ossec.log | tail -200"
```

Les lignes importantes observées :

```
Directory set for real time monitoring: '/opt'.
Directory set for real time monitoring: '/tmp'.
Directory set for real time monitoring: '/var/www'.
File integrity monitoring scan ended.
```

```
FIM sync module started.
Real-time file integrity monitoring started.
```

Ce test est important : une configuration XML présente sur le disque ne prouve pas que wazuh-syscheckd l'a chargée.

7. Script Active Response YARA

Le script est stocké dans le rôle Ansible puis copié sur les agents dans /var/ossec/active-response/bin/yara.sh. Il doit être exécutable par le mécanisme Wazuh.

```
- name: Deployer le script Active Response YARA
  ansible.builtin.copy:
    src: yara.sh
    dest: /var/ossec/active-response/bin/yara.sh
    owner: root
    group: wazuh
    mode: "0750"
```

Version robuste du script validé :

```
#!/bin/bash

LOG_FILE="/var/ossec/logs/active-responses.log"
YARA="/usr/bin/yara"
YARA_RULES="/opt/yara/packages/core/yara-rules-core.yar"
QUARANTINE_DIR="/var/quarantine"

read -r INPUT_JSON

COMMAND=$(echo "$INPUT_JSON" | jq -r '.command // empty')
if [ "$COMMAND" != "add" ]; then
    exit 0
fi

FILE_PATH=$(echo "$INPUT_JSON" | jq -r '.parameters.alert.syscheck.path // empty')

if [ -z "$FILE_PATH" ]; then
    echo "wazuh-yara: ERROR - Aucun chemin recu" >> "$LOG_FILE"
    exit 1
fi

case "$FILE_PATH" in
    /var/quarantine/*)
        exit 0
        ;;
esac

if [ ! -f "$FILE_PATH" ]; then
    echo "wazuh-yara: INFO - Fichier absent ou deja supprime: $FILE_PATH" >> "$LOG_FILE"
    exit 0
fi

YARA_OUTPUT=$("$YARA" -w "$YARA_RULES" "$FILE_PATH" 2>>"$LOG_FILE")
YARA_RC=$?

if [ "$YARA_RC" -ne 0 ]; then
    echo "wazuh-yara: ERROR - Erreur de scan YARA pour: $FILE_PATH (rc=$YARA_RC)" >> "$LOG_FILE"
    exit 0
fi

if [ -z "$YARA_OUTPUT" ]; then
    echo "wazuh-yara: INFO - Aucun match: $FILE_PATH" >> "$LOG_FILE"
    exit 0
fi

while IFS= read -r line; do
    [ -n "$line" ] && echo "wazuh-yara: ALERT - Scan result: $line" >> "$LOG_FILE"
done <<< "$YARA_OUTPUT"
```

```
TIMESTAMP=$(date +"%Y%m%d_%H%M%S")
BASENAME=$(basename "$FILE_PATH")
QUARANTINE_PATH="${QUARANTINE_DIR}/${TIMESTAMP}_${BASENAME}"

mkdir -p "$QUARANTINE_DIR"
chmod 0700 "$QUARANTINE_DIR"

if mv -- "$FILE_PATH" "$QUARANTINE_PATH"; then
    chmod 000 "$QUARANTINE_PATH"
    echo "wazuh-yara: ALERT - Fichier mis en quarantaine" >> "$LOG_FILE"
    echo "wazuh-yara: ALERT - Original: $FILE_PATH" >> "$LOG_FILE"
    echo "wazuh-yara: ALERT - Quarantaine: $QUARANTINE_PATH" >> "$LOG_FILE"
else
    echo "wazuh-yara: ERROR - Echec de mise en quarantaine: $FILE_PATH" >> "$LOG_FILE"
    exit 1
fi

exit 0
```

Sécurité critique — Ne jamais interpréter stderr comme un match. Une ancienne version mélangeait les erreurs YARA avec la sortie normale : 'error scanning ... could not open file' était alors pris pour une détection. On sépare stderr et on vérifie YARA_RC avant toute quarantaine.

8. Déclarer la commande et l'Active Response sur le manager

Dans /var/ossec/etc/ossec.conf du Wazuh Manager :

```
<command>
  <name>yara-scan</name>
  <executable>yara.sh</executable>
  <timeout_allowed>no</timeout_allowed>
</command>

<active-response>
  <disabled>no</disabled>
  <command>yara-scan</command>
  <location>local</location>
  <rules_id>550,554</rules_id>
</active-response>
```

La règle 554 correspond à l'ajout d'un fichier dans FIM et 550 à une modification de checksum. location=local signifie que l'action est exécutée sur l'agent qui a produit l'événement.

Attention — Une Active Response mal ciblée peut supprimer ou déplacer des fichiers légitimes. Le choix 'quarantaine' est volontairement préférable à une suppression définitive.

9. Tester YARA avant de tester Wazuh

Avant d'accuser Active Response, il faut vérifier que le moteur YARA détecte réellement le fichier.

```
yara -w /opt/yara/packages/core/yara-rules-core.yar /opt/eicar.com
```

Résultat obtenu :

```
TRELLIX_ARC_Malw_Eicar /opt/eicar.com
```

EICAR est un fichier de test standard destiné à vérifier le fonctionnement d'un moteur de détection. Il permet de valider la chaîne sans utiliser un malware réel.

Le ruleset contenait bien une règle EICAR, retrouvée avec :

```
grep -in 'eicar' /opt/yara/packages/core/yara-rules-core.yar | head -20
```

10. Test réel de quarantaine

```
ansible client01 -m shell -a "cp /opt/eicar.com /opt/eicar-QUARANTINE-V3.bin"

ansible client01 -m shell -a "tail -n 20 /var/ossec/logs/active-responses.log"

ansible client01 -m shell -a "ls -l /var/quarantine | tail"
```

Résultat validé :

```
wazuh-yara: ALERT - Scan result: TRELLIX_ARC_Malw_Eicar /opt/eicar-QUARANTINE-V3.bin
wazuh-yara: ALERT - Fichier mis en quarantaine
wazuh-yara: ALERT - Original: /opt/eicar-QUARANTINE-V3.bin
wazuh-yara: ALERT - Quarantaine: /var/quarantine/20260816_023319_eicar-QUARANTINE-V3.bin
```

Et dans le répertoire :

```
---------- 1 root root 68 Aug 16 02:33 20260816_023319_eicar-QUARANTINE-V3.bin
```

Interprétation de chmod 000 — Le fichier n'a plus de droits Unix classiques de lecture, écriture ou exécution. Root garde néanmoins la capacité administrative de modifier les permissions ou de récupérer le fichier.

11. Faire apparaître les actions YARA dans le Dashboard

La quarantaine fonctionnait déjà dans active-responses.log, mais rien n'était clairement visible dans Threat Hunting. La solution consiste à décoder les lignes personnalisées puis à créer des règles Wazuh dédiées.

11.1 Decoders

Dans /var/ossec/etc/decoders/local_decoder.xml :

```
<decoder name="yara_scan">
  <prematch>^wazuh-yara: ALERT - Scan result: </prematch>
  <regex offset="after_prematch">(\S+) (\S+)</regex>
  <order>yara_rule,yara_scanned_file</order>
</decoder>

<decoder name="yara_quarantine">
  <prematch>^wazuh-yara: ALERT - Quarantaine:</prematch>
  <regex type="pcre2">Quarantaine:\s+(.+)$</regex>
  <order>quarantine_path</order>
</decoder>
```

11.2 Règles

Dans /var/ossec/etc/rules/local_rules.xml :

```
<group name="yara,malware,active_response,">

  <rule id="100300" level="12">
    <decoded_as>yara_scan</decoded_as>
    <description>YARA malware detected: $(yara_rule) on $(yara_scanned_file)</description>
  </rule>

  <rule id="100301" level="10">
    <decoded_as>yara_quarantine</decoded_as>
    <description>YARA file quarantined: $(quarantine_path)</description>
  </rule>

</group>
```

Les deux alertes ont des rôles différents :

- 100300, niveau 12 : YARA a identifié une règle malveillante sur un fichier.
- 100301, niveau 10 : l'action de quarantaine a effectivement été journalisée.

12. Valider les decoders avec wazuh-logtest

```
/var/ossec/bin/wazuh-logtest
```

Test de détection :

```
wazuh-yara: ALERT - Scan result: TRELLIX_ARC_Malw_Eicar /opt/eicar-QUARANTINE-V3.bin
```

```
Starting wazuh-logtest v4.14.5
Type one log per line

wazuh-yara: ALERT - Scan result: TRELLIX_ARC_Malw_Eicar /opt/eicar-QUARANTINE-V3.bin

**Phase 1: Completed pre-decoding.
        full event: 'wazuh-yara: ALERT - Scan result: TRELLIX_ARC_Malw_Eicar /opt/eicar-QUARANTINE-V3.bin'

**Phase 2: Completed decoding.
        name: 'yara_decoder'
        yara_rule: 'TRELLIX_ARC_Malw_Eicar'
        yara_scanned_file: '/opt/eicar-QUARANTINE-V3.bin'
```

Premier test : le decoder de scan extrait correctement yara_rule et yara_scanned_file.

Test de quarantaine :

```
wazuh-yara: ALERT - Quarantaine: /var/quarantine/20260816_023319_eicar-QUARANTINE-V3.bin
```

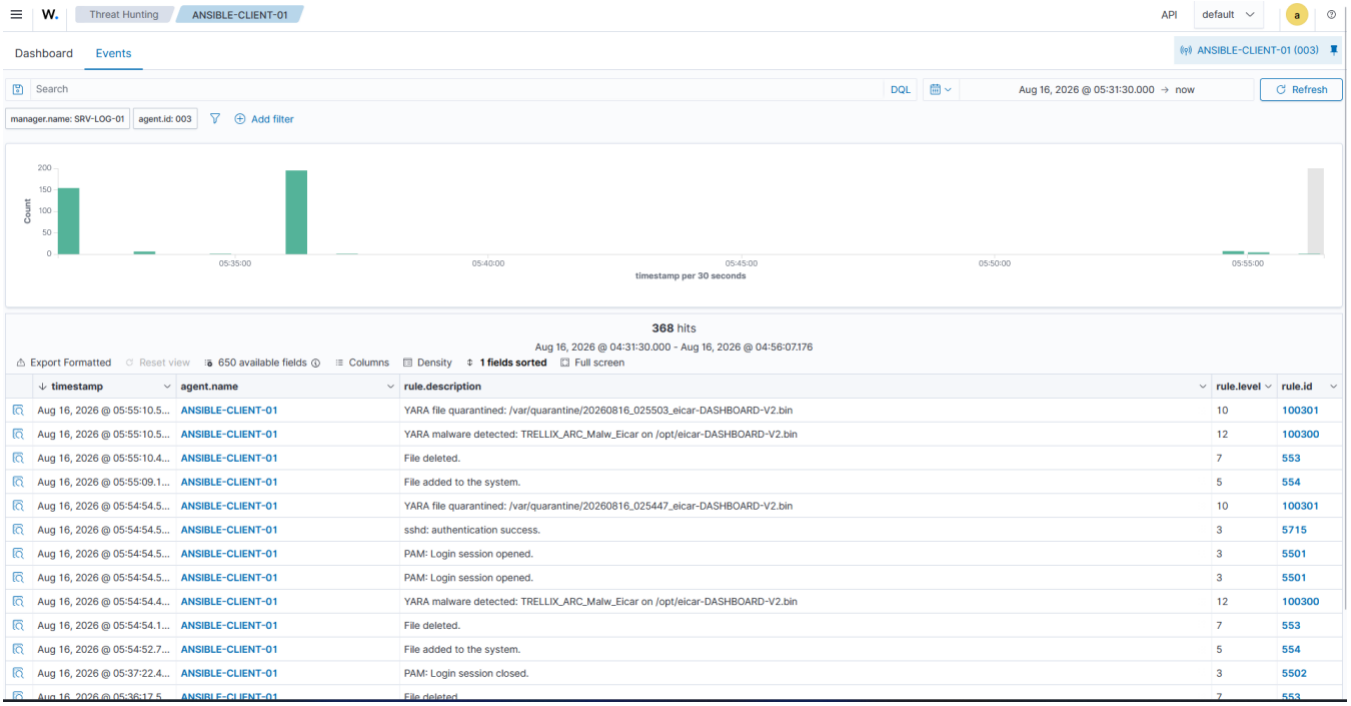
Résultat final attendu :

```
Phase 2:
  name: 'yara_quarantine'
  quarantine_path: '/var/quarantine/20260816_023319_eicar-QUARANTINE-V3.bin'

Phase 3:
  id: '100301'
  level: '10'
  description: 'YARA file quarantined: /var/quarantine/...'
```

Leçon de dépannage — Un decoder peut être reconnu par son nom sans extraire le champ attendu. Il faut vérifier la Phase 2, pas seulement la Phase 3. Ici, le passage à une regex PCRE2 explicite a corrigé quarantine_path.

13. Résultat dans Threat Hunting



Dashboard final : la détection YARA et la quarantaine sont visibles comme deux alertes distinctes.

Sur la capture, la séquence est lisible directement :

1. Rule 554 — File added to the system.
2. Rule 100300 — YARA malware detected: TRELLIX_ARC_Malw_Eicar.
3. Rule 553 — File deleted de son emplacement original, conséquence du déplacement.
4. Rule 100301 — YARA file quarantined avec le chemin final.

Résultat — Le SOC n'a plus besoin de consulter manuellement active-responses.log : la détection et la réponse apparaissent dans l'interface Wazuh.

14. Health check post-déploiement

Une erreur importante rencontrée pendant le projet : systemctl pouvait indiquer que wazuh-agent était actif alors que plusieurs démons internes ne tournaient plus. Le playbook vérifie donc directement wazuh-control status.

```
post_tasks:

 - name: Verifier les composants internes Wazuh
   ansible.builtin.command:
     cmd: /var/ossec/bin/wazuh-control status
   register: wazuh_status
   changed_when: false
   failed_when: >
     'wazuh-modulesd is running' not in wazuh_status.stdout or
     'wazuh-logcollector is running' not in wazuh_status.stdout or
     'wazuh-syscheckd is running' not in wazuh_status.stdout or
     'wazuh-agentd is running' not in wazuh_status.stdout or
     'wazuh-execd is running' not in wazuh_status.stdout
```

```
  - name: Afficher la sante Wazuh
    ansible.builtin.debug:
      var: wazuh_status.stdout_lines
```

État final du pilote :

```
wazuh-modulesd is running...
wazuh-logcollector is running...
wazuh-syscheckd is running...
wazuh-agentd is running...
wazuh-execd is running...

PLAY RECAP
client01 : ok=18 changed=0 unreachable=0 failed=0
```

Idempotence — changed=0 lors d'un second passage signifie que la baseline est déjà dans l'état demandé : Ansible n'a rien eu à modifier.

15. Erreurs rencontrées et méthode de correction

Symptôme	Cause	Correction
YARA test-rule.yar : syntax error	Le caractère $ avait été écrit littéralement avec un antislash dans le fichier.	Créer une vraie règle multi-ligne : $a = "..." puis condition: $a.
Aucun événement FIM	wazuh-syscheckd n'avait pas encore correctement chargé/démarré la surveillance.	Contrôler ossec.log et rechercher 'Real-time file integrity monitoring started'.
error scanning ... could not open file	Fichiers temporaires Ansible/APT supprimés avant le passage de YARA.	Ignorer les chemins temporaires bruyants et vérifier l'existence du fichier avant le scan.
Erreur YARA interprétée comme malware	stderr avait été fusionné avec la sortie de match.	Rediriger stderr séparément et vérifier YARA_RC avant de considérer YARA_OUTPUT.
Active Response fonctionne mais rien dans le Dashboard	Les logs custom n'avaient ni decoder ni règle.	Créer local_decoder.xml + local_rules.xml puis tester avec wazuh-logtest.
Description 'YARA file quarantined:' vide	Le decoder matchait mais n'extrayait pas quarantine_path.	Regex PCRE2 : Quarantaine:\s+(.+)$ avec <order>quarantine_path</order>.
Script modifié mais comportement inchangé	Le fichier du rôle Ansible avait été édité mais pas redéployé.	Relancer baseline.yml --limit client01 et vérifier YARA_RC dans le script distant.
systemctl wazuh-agent actif mais FIM/execd morts	Le service parent ne suffisait pas à prouver l'état des composants internes.	Utiliser /var/ossec/bin/wazuh-control status et un failed_when Ansible.

16. Commandes de diagnostic à retenir

```
# Syntaxe Ansible
ansible-playbook /etc/ansible/playbooks/baseline.yml --syntax-check

# Déploiement pilote
ansible-playbook /etc/ansible/playbooks/baseline.yml --limit client01

# Santé Wazuh
ansible client01 -m shell -a "/var/ossec/bin/wazuh-control status"

# Logs FIM
ansible client01 -m shell -a "grep -iE 'syscheck|scan|realtime|inotify' /var/ossec/logs/ossec.log | tail -200"

# Logs Active Response
ansible client01 -m shell -a "tail -n 50 /var/ossec/logs/active-responses.log"

# Test YARA direct
ansible client01 -m shell -a "yara -w /opt/yara/packages/core/yara-rules-core.yar /opt/eicar.com"

# Quarantaine
ansible client01 -m shell -a "ls -l /var/quarantine"

# Test des decoders/règles sur le manager
/var/ossec/bin/wazuh-logtest
```

17. Checklist de reconstruction

5. Installer yara, jq et unzip sur l'agent.
6. Créer /opt/yara et /var/quarantine.
7. Télécharger et extraire YARA Forge Core.
8. Vérifier manuellement le ruleset avec un fichier sain puis EICAR.

9. Configurer FIM et ses exclusions.
10. Vérifier dans ossec.log que le monitoring temps réel est démarré.
11. Déployer yara.sh dans active-response/bin avec les bons droits.
12. Déclarer <command> et <active-response> sur le manager.
13. Tester un EICAR et vérifier active-responses.log + /var/quarantine.
14. Créer yara_scan et yara_quarantine dans local_decoder.xml.
15. Créer les règles 100300 et 100301 dans local_rules.xml.
16. Valider les deux lignes avec wazuh-logtest.
17. Redémarrer wazuh-manager après validation.
18. Refaire un test EICAR réel et vérifier Threat Hunting.
19. Ajouter le health check Wazuh au playbook.
20. Déployer progressivement sur la flotte.

18. Points de sécurité et limites

- Une règle YARA peut produire un faux positif. La quarantaine est donc plus prudente qu'une suppression automatique.
- chmod 000 n'est pas une barrière contre root : c'est un confinement par permissions Unix classiques.
- Le répertoire /var/quarantine doit rester exclu de FIM pour éviter les boucles.
- Le ruleset YARA doit être mis à jour de façon contrôlée et testé avant généralisation.
- Surveiller trop de chemins en realtime peut créer du bruit et consommer des ressources. Les exclusions doivent être adaptées au rôle du serveur.
- Les événements FIM 550/554 sont larges : la logique de sécurité finale dépend donc fortement de la robustesse de yara.sh.
- Le pilote doit toujours être validé avant déploiement massif.

19. Synthèse technique

Cette réalisation montre une chaîne complète de détection et réponse automatisée : administration de flotte avec Ansible, surveillance d'intégrité avec Wazuh FIM, moteur de signatures YARA, règles communautaires YARA Forge, Active Response, quarantaine contrôlée, décodage de logs personnalisé, règles SIEM custom et visualisation dans Threat Hunting. L'intérêt du projet n'est pas seulement l'installation des outils, mais leur intégration, leur dépannage et leur automatisation.

20. Résumé à mémoriser

```
FIM voit le changement.
YARA analyse le contenu.
Active Response orchestre l'action.
yara.sh décide : sain / erreur / match.
Match -> mv vers /var/quarantine + chmod 000.
Decoder transforme le log custom en champs.
Rule 100300 = malware détecté.
Rule 100301 = quarantaine confirmée.
Threat Hunting rend le tout visible.
Ansible rend la configuration reproductible.
```

Synthèse — Cette configuration associe Wazuh FIM, YARA Forge et Active Response afin d'analyser automatiquement les fichiers détectés, isoler les correspondances malveillantes et afficher la détection ainsi que la quarantaine dans Wazuh.

Réalisé par Adel Sadek