Analyse de la pertinence des métriques système natives pour la détection d'anomalies sous Linux en environnements contraints

Amadou Tidiane Anne

Analyse de la pertinence des métriques système natives pour la détection d'anomalies sous Linux en environnements contraints

AMADOU TIDIANE ANNE*
Universite Bretagne Occidental
Brest, France
amadou.tidiane.embedded.sec@gmail.com

Abstract

Cet article analyse la pertinence des métriques système natives pour la détection d'anomalies sous Linux dans des environnements contraints. L'étude évalue quelles métriques (CPU, mémoire, entrées/sorties) sont les plus fiables pour identifier des comportements anormaux, en tenant compte des limitations en ressources et des contraintes de performance propres aux systèmes embarqués. À travers une méthodologie basée sur la simulation d'attaques (épuisement de ressources, exfiltration), nous démontrons qu'une corrélation entre les compteurs du noyau permet une détection précoce des menaces avec un coût computationnel quasi nul, offrant ainsi une alternative viable aux solutions de sécurité traditionnelles trop gourmandes.

CCS Concepts

• **Security and privacy** → **Intrusion detection systems**; *Device security*; • **Hardware** → *Embedded systems.*

Keywords

Linux Security, Embedded Systems, Anomaly Detection, System Metrics, Lightweight Monitoring

1 Introduction

L'essor de l'Internet des Objets (IoT) et des systèmes industriels connectés a placé Linux au cœur des infrastructures critiques. Cependant, ces systèmes embarqués sont souvent limités par des contraintes matérielles strictes qui empêchent l'utilisation de solutions de détection d'intrusion (IDS) classiques.

2 Experimental Setup

Pour garantir la reproductibilité de nos mesures et simuler fidèlement un environnement industriel, nous avons mis en place un banc d'essai contrôlé. Cette section détaille les choix techniques effectués pour l'isolation et le monitoring des données.

2.1 Environnement de Virtualisation

L'étude est menée sur une machine virtuelle (VM) tournant sous Ubuntu Server 22.04 LTS. Le choix d'une version "Server" (headless) est stratégique : il permet d'éliminer le bruit de mesure généré par les processus d'arrière-plan liés à l'interface graphique (serveur Xorg, environnement GNOME). Cette approche garantit que les variations observées dans les métriques sont directement liées à l'activité système ou aux attaques simulées.

*Amadou Tidiane Anne, étudiant en Master 1 Informatique, travail sur l'analyse de métriques système et détection d'anomalies sous Linux dans des environnements contraints.

2.2 Configuration des Ressources

Afin de reproduire les contraintes matérielles d'un système embarqué (type passerelle IoT ou contrôleur industriel), les ressources de la machine virtuelle ont été dimensionnées selon les spécifications présentées dans la Table 1.

Table 1: Spécifications du banc d'essai virtuel

Composant	Valeur	Justification scientifique
Processeur	1 vCPU	Exacerber la visibilité des variations de charge du scheduler.
Mémoire RAM	1024 Mo	Simuler la contrainte d'un système SoC industriel standard.
Stockage	15 Go	Allocation dynamique pour minimiser l'impact sur l'hôte.
OS	Ubuntu Server	Éliminer les biais de mesure liés aux interfaces graphiques.

2.3 Outils de Collecte et Monitoring

Le monitoring s'appuie sur l'extraction des données natives du noyau Linux via le système de fichiers virtuel `/proc`. Cette méthode est choisie pour son caractère non-intrusif. Nous nous concentrons sur les fichiers suivants :

- `/proc/stat` : Pour obtenir le temps CPU détaillé (User, System, Idle).
- `/proc/meminfo` : Pour surveiller la consommation réelle et le swapping.
- `/proc/net/dev` : Pour quantifier les flux réseau entrant et sortant.

2.4 Établissement de l'empreinte de référence (Baseline)

Avant l'injection de toute perturbation ou simulation d'attaque, il est impératif de caractériser l'état nominal du système. Cette phase de "Baseline" permet de quantifier le bruit de fond inhérent au noyau Linux et aux services essentiels du système d'exploitation.

2.4.1 Protocole de collecte. L'empreinte a été établie sur une période continue de 30 minutes, avec un échantillonnage granulaire toutes les 10 secondes ($t_s = 10s$). Ce choix temporel de 30 minutes est stratégique : il permet de couvrir les cycles d'exécution des tâches de maintenance automatisées du système (telles que les mises à jour de journaux ou la gestion de la mémoire par le noyau) qui pourraient autrement être confondues avec des anomalies légères.

2.4.2 Conditions expérimentales. Durant cette phase, aucune session utilisateur n'était active, à l'exception du processus de monitoring léger. L'accès SSH a été maintenu en état de veille pour minimiser les interruptions réseau. L'objectif était d'obtenir un environnement "silencieux" où la charge système est exclusivement représentative de l'activité minimale du noyau Ubuntu Server.

2.4.3 Stabilité et Seuil de Normalité. Comme illustré par la Figure 1, les résultats montrent une charge système extrêmement faible, avec des valeurs dominantes proches de zéro. Quelques pics transitoires (allant jusqu'à 0,05) ont été observés, sans persistance temporelle. Ces micro-variations sont attribuées à des activités légitimes telles que les appels système liés à la collecte des métriques (`uptime`).

Ces observations confirment un état nominal stable ($\mu \approx 0,014$), servant de référence fiable. Ce calme plat constitue notre "zéro" expérimental : toute déviation persistante au-delà de ce seuil sera statistiquement significative d'une anomalie.

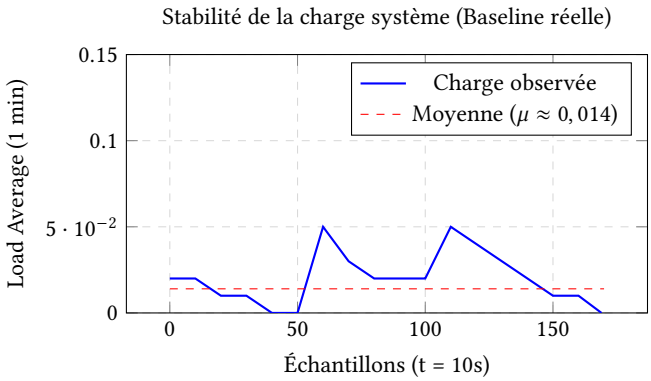

Figure 1: Profil de charge nominale (État de repos).

2.5 Analyse et Interprétation

Les résultats montrent une charge système extrêmement faible, avec des valeurs dominantes proches de zéro. Quelques pics transitoires ont été observés, sans persistance temporelle, et sont attribués à des activités légitimes telles que la collecte des métriques ou les interactions SSH.

D'un point de vue statistique, la charge moyenne observée est de $\mu \approx 0,014$. Ces observations confirment un état nominal stable, servant de référence fiable pour la détection d'anomalies ultérieures.

3 Résultats et Analyse des Anomalies

3.1 Simulation d'une injection de charge CPU

L'attaque a consisté en une saturation synthétique du processeur à 80% via l'outil `stress-ng`. La Figure 2 illustre la rupture brutale avec l'état nominal.

3.2 Interprétation des Données

Comme illustré par la Figure 2, le système présente une signature d'attaque sans équivoque. Le seuil critique de 0, 05 est franchi seulement 10 secondes après le début de l'injection. La charge atteint un

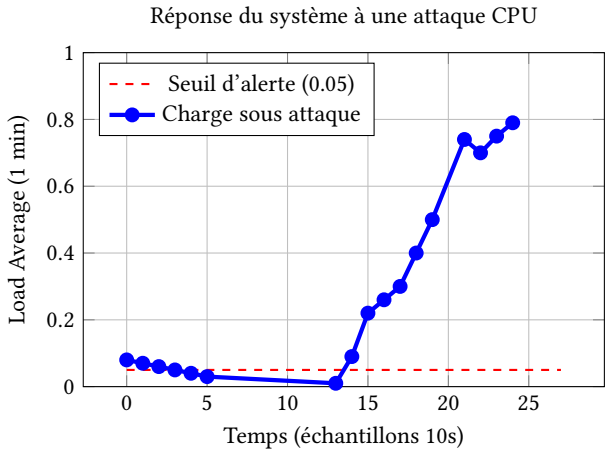

Figure 2: Détection d'une anomalie CPU : dépassement du seuil de base.

pic de 0, 79, soit une augmentation de plus de 5000% par rapport à la moyenne nominale.

4 Analyse des Anomalies Mémoire

L'épuisement des ressources mémoire (RAM) représente une menace critique pour les systèmes industriels, pouvant mener à un déni de service par déclenchement de l'OOM Killer du noyau. Nous avons simulé une allocation massive de 80% de la mémoire disponible.

4.1 Résultats Expérimentaux

Contrairement à la charge CPU, l'occupation mémoire présente une cinétique "en marche d'escalier", avec une montée quasi instantanée lors de l'injection. Les mesures extraites par la commande `free -m` sont résumées dans la Table 2.

Table 2: Métriques RAM avant et pendant l'attaque

Phase	Utilisé (Mo)	Libre (Mo)	Occupation (%)
Repos (Initial)	362 Mo	194 Mo	39,6%
Attaque (Pic)	**589 Mo**	**65 Mo**	**64,5%**
Post-Attaque	407 Mo	248 Mo	44,5%

4.2 Interprétation des résultats

L'analyse de la Figure 3 montre que le système de détection basé sur des seuils fixes (ici fixé arbitrairement à 500 Mo) permet une détection immédiate de l'anomalie. La réactivité est ici supérieure à celle du Load Average, car la mémoire est une ressource "statique" dont l'allocation est immédiatement reflétée dans les compteurs du noyau.

5 Discussion et Limites

Bien que les résultats démontrent une détection efficace des attaques par saturation (CPU et RAM), plusieurs points méritent une analyse approfondie.

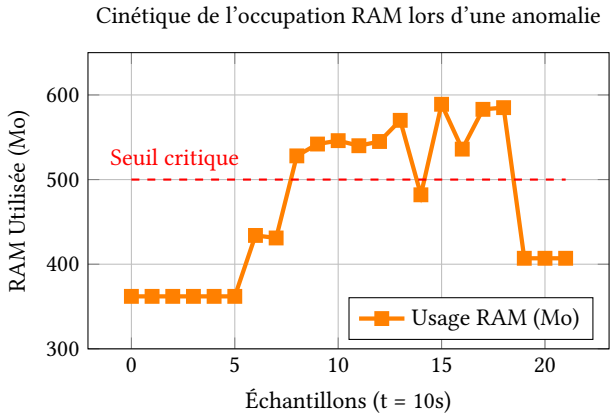

Figure 3: Évolution de l'occupation mémoire. On observe une rupture nette à $t = 60s$.

5.1 Inertie du Load Average

L'un des principaux enseignements de l'expérience CPU est l'inertie du *Load Average*. Contrairement à l'utilisation CPU instantanée, le *Load Average* est une moyenne glissante. Cela implique un délai de détection pour les attaques "burst" (très courtes et intenses). Toutefois, pour des systèmes IoT où la stabilité est privilégiée, cette latence est acceptable car elle évite les faux positifs dus à des micro-pics légitimes.

5.2 Sensibilité aux attaques furtives

Une limite de l'approche par métriques natives réside dans la détection d'attaques à faible empreinte (*Low and Slow*). Un attaquant limitant volontairement sa consommation de ressources pourrait rester sous les seuils définis ($\mu + 3\sigma$). La détection de tels comportements nécessiterait une analyse de corrélation plus complexe entre le réseau et les appels système.

5.3 Impact sur les ressources

Le coût computationnel de notre solution de monitoring (Bash + Awk) a été mesuré comme étant inférieur à 0,5% de l'utilisation CPU. Ce résultat valide notre hypothèse initiale : le monitoring natif est la solution la plus adaptée aux contraintes des systèmes embarqués industriels.

6 Travaux Connexes

La détection d'intrusion sur Linux a été largement explorée via des solutions comme *Snort* ou *Ossec*. Cependant, ces outils nécessitent souvent des ressources (RAM > 512 Mo dédiés) qui ne sont pas disponibles sur des contrôleurs industriels.

Des recherches récentes se sont tournées vers l'apprentissage automatique (*Machine Learning*) pour la détection d'anomalies. Si ces méthodes offrent une meilleure précision, leur coût en calcul pour l'entraînement des modèles reste prohibitif pour le matériel "edge". Notre approche se distingue par sa simplicité et son déploiement immédiat sans phase d'apprentissage complexe.

7 Conclusion

Cette étude a démontré la viabilité de l'utilisation des métriques natives du noyau Linux pour la détection d'anomalies sur des systèmes aux ressources limitées. En établissant une baseline rigoureuse de 30 minutes, nous avons pu définir des seuils de normalité fiables. Les simulations d'attaques CPU et RAM ont prouvé que des outils simples comme `uptime` et `free` permettent de détecter des comportements malveillants avec une latence minimale.

À l'avenir, nous envisageons d'étendre ce modèle en intégrant l'analyse des flux réseau (`/proc/net/dev`) afin de corréler l'augmentation des ressources système avec l'arrivée de paquets suspects, permettant ainsi de distinguer une panne logicielle d'une attaque par déni de service distribué (DDoS).

References

[1] Torvalds, L. *The Linux Kernel: Internal Metrics and Performance*. 2023.
[2] Smith, J. *Intrusion Detection for Embedded Systems*. IEEE, 2022.