UNIVERSITÉ DE LORRAINE | IUT nancy Charlemagne

BUT informatique 3A, parcours DACS

SAÉ de projet tuteuré

Analyse forensique et débogage système
Janvier 2026

1 Objectif

L'objectif de ce projet est de développer des compétences approfondies en diagnostic et résolution de problèmes sur des systèmes Linux, en utilisant intensivement les outils et commandes standards d'analyse.

Ce projet mettra l'accent sur :
- la maîtrise d'outils de diagnostic système avancés,
- l'analyse de comportements anormaux et la recherche de causes profondes,
- l'investigation de traces d'exécution et de logs système,
- la documentation méthodique de démarches d'analyse.

Le travail s'effectue par groupe de 3 étudiants. L'environnement Linux est **obligatoire** pour les travaux demandés.

L'évaluation finale se composera du rendu des ressources demandées ainsi qu'une mini-soutenance pour répondre du travail réalisé et démontrer les compétences acquises en diagnostic système.

2 Descriptif des étapes à réaliser

1 Analyse de processus et performances

Objectif : diagnostiquer des problèmes de performances et analyser le comportement des processus.

Dans cette partie, il s'agit de développer des compétences avec les outils suivants :
- `ps`, `top`, `htop` : identifier les processus consommateurs de ressources, analyser les relations parent/enfant, comprendre les états des processus (R, S, D, Z, T),
- `pstree`, `pgrep`, `pidof` : naviguer dans l'arbre des processus, retrouver des processus par nom ou critères,
- `lsof` : lister les fichiers ouverts par un processus, identifier les processus utilisant un fichier ou un port réseau,
- `strace` : tracer les appels système d'un processus, comprendre son comportement, identifier des erreurs,
- `/proc` : explorer le système de fichiers `/proc` pour obtenir des informations détaillées sur les processus.

Travaux demandés :

1. Créer plusieurs scénarios de test avec des processus présentant des comportements particuliers (ex. : boucle infinie, fuite mémoire, blocage I/O).
2. Pour chaque scénario, documenter la méthodologie de diagnostic et les commandes utilisées pour identifier et comprendre le problème.
3. Développer un script Ruby générant un rapport d'analyse de processus incluant : PID, nom, état, consommation CPU/mémoire, fichiers ouverts, appels système récents.

❷ Diagnostic réseau

Objectif : analyser et diagnostiquer des problèmes de connectivité et de performance réseau.

Dans cette partie, il s'agit de maîtriser les outils suivants :
— `ss`, `netstat` : lister les sockets actives, identifier les connexions établies, les ports en écoute,
— `ip` (remplaçant `ifconfig`) : configuration et inspection des interfaces réseau, tables de routage, règles de politique,
— `tcpdump` : capturer et analyser le trafic réseau, filtrer par protocole/port/hôte,
— `nmap` : scanner les ports ouverts, identifier les services actifs,
— `traceroute`, `mtr` : analyser le chemin réseau et identifier les points de latence,
— `dig`, `nslookup`, `host` : diagnostiquer les problèmes DNS.

Travaux demandés :

1. Créer des scénarios de problèmes réseau (ex. : service inaccessible, latence élevée, résolution DNS défaillante, pare-feu bloquant).
2. Pour chaque scénario, documenter la démarche complète de diagnostic avec les commandes utilisées et leur interprétation.
3. Développer un script d'audit réseau automatisé identifiant :
   — les services exposés sur le réseau,
   — les connexions suspectes ou non attendues,
   — les problèmes de configuration réseau.

❸ Analyse de logs et investigation post-incident

Objectif : mener une investigation forensique après un incident système simulé.

Dans cette partie, il s'agit de :

1. Mettre en place un scénario d'incident simulé (ex. : compromission d'un compte, arrêt inopiné d'un service, saturation disque, tentative d'intrusion).
2. Mener une investigation complète en utilisant :
   — `journalctl` : analyser les logs systemd, filtrer par unité, par priorité, par période,
   — `last`, `lastlog`, `who`, `w` : identifier les connexions utilisateurs,
   — `ausearch`, `aureport` : exploiter les logs d'audit système (si auditd est configuré),
   — analyse manuelle des logs dans `/var/log` (`auth.log`, `syslog`, etc.),
   — `grep`, `awk`, `sed` : extraction et corrélation d'informations depuis les logs.
3. Reconstituer la chronologie de l'incident : quoi, quand, qui, comment.
4. Identifier les traces laissées par l'incident et les moyens de détection.

Travaux demandés :
— Créer au moins 2 scénarios d'incident différents.
— Pour chaque scénario, produire un rapport d'investigation forensique structuré (contexte, méthodologie, découvertes, chronologie, recommandations).
— Développer un script d'analyse automatique de logs permettant de détecter des patterns suspects.

❹ Outils de diagnostic système avancés

Objectif : explorer des outils de diagnostic plus spécialisés.

Dans cette partie, il s'agit d'expérimenter avec :
— `vmstat`, `iostat`, `mpstat` : statistiques système (mémoire, I/O, CPU),
— `iotop` : identifier les processus consommateurs d'I/O disque,
— `sar` : collecter et analyser l'historique des performances système,
— `dmesg` : analyser les messages du noyau, identifier des erreurs matérielles,

— `systemd-analyze` : analyser les temps de démarrage du système et des services.

Travaux demandés :

1. Créer un tableau de bord de diagnostic système (format texte ou HTML) agrégeant les informations de tous ces outils.
2. Identifier et documenter un problème de performance système réel ou simulé en utilisant ces outils.

5 Automatisation et conteneurisation

Objectif : packager les outils d'analyse développés.

Dans cette partie, il s'agit de :

1. Développer un script Ruby orchestrant tous les scripts d'analyse développés.
2. Créer un `Dockerfile` intégrant tous les outils nécessaires.
3. Documenter l'utilisation de la solution dans différents contextes (analyse en temps réel, post-mortem, audit de routine).

3 Résultats attendus

— Un dépôt Git avec tous les scripts développés (shell et Ruby).
— Les scénarios de test et d'incident créés (scripts de reproduction).
— Un mini-rapport comprenant :
  — un guide de référence des commandes étudiées (usage, cas d'application, exemples),
  — les rapports d'investigation des incidents simulés,
  — une réflexion sur les méthodologies de diagnostic et les bonnes pratiques,
— Le `Dockerfile` et la documentation d'utilisation.
— Une démonstration en direct :
  — diagnostic d'un problème système préparé par l'enseignant,
  — présentation des outils et méthodes maîtrisés,
  — capacité à expliquer et interpréter les sorties des commandes.